1964

A digital differential analyzer programming system for the IBM 7074 computer

George Joseph Farris
Iowa State University

A DIGITAL DIFFERENTIAL ANALYZER PROGRAMMING SYSTEM FOR THE I.B.M. 7074 COMPUTER

by

George Joseph Farris

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major Subject: Chemical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

Head of Major Department

Signature was redacted for privacy.

Dean of Graduate College

Iowa State University
Of Science and Technology
Ames, Iowa

1964

TABLE OF CONTENTS

I. INTRODUCTION

The purpose of this project was to develop a system for programming the I.B.M. 7074 general purpose digital computer in terms of individual computational components similar to the components of analog computers and conventional digital differential analyzers.

In many problems arising in engineering, particularly in the simulation of processes and equipment, it may be faster, simpler, and more meaningful to use an analog method of solution than to reduce the problem to its most compact mathematical form and write a digital computer program for solving the resulting equations.

With the analog method, equipment and processes are simulated by individual computational building blocks, each of which performs a specific mathematical operation and usually corresponds to a specific part of the physical system.

In programming an analog device a diagram is prepared indicating the components required and their interconnection. This block diagram method makes analog programming relatively simple and straightforward. The individual computer components are then physically connected to form an analog of the problem.

The most widely used analog device is the electronic differential analyzer or analog computer which, while being a convenient computational device, has several limitations, especially for large problems. Large analog computers are

not as readily available as large digital computers. Also, there is a limit to the accuracy obtainable with analog computers and accuracy decreases as the problem size is increased, especially as the number of multiplications, divisions, and non-linear functions required becomes large.

Except for analog computers equipped with special storage devices, pure transport delay is difficult to achieve without distortion. Also, since the allowable voltage range of the electronic components is limited, scaling is required with all analog computers.

A second type of analog device is the digital differential analyzer which carries out the mathematical operations digitally rather than continuously. These machines are more accurate than analog computers but they are slower and more expensive. Their advantage over general purpose digital computers is that they use the analog programming method.

This project has resulted in a system for programming a general purpose digital computer (specifically the I.B.M. 7074) in a manner similar to the method of programming analog devices. In using this system the programer need only be concerned with the relationship between inputs and outputs of basic computer components. He need not be concerned with the digital computer programming necessary to accomplish the mathematical operations which these components perform or the mechanics of interconnecting them. The system eliminates

the above disadvantages of analog computers and is more convenient to use than a conventional digital differential analyzer. Because it combines some of the advantages of DIgital and ANalog computers, the system has been named DIAN.

Two versions of the system have been developed, one designed for use with an I.B.M. 7070 or 7074 computer restricted to a 10,000 word core storage (referred to as the 10K system), and one for use on the same type of machine but with a 20,000 word core (referred to as the 20K system). Both require two work tapes in addition to the input and output tapes.

II. PREVIOUS WORK

The term "differential analyzer" was originally applied to devices used primarily for solving differential equations. The first scientifically useful differential analyzer was designed by Bush and Caldwell (1) and was widely used during World War II. Its major components were eighteen ball and disk mechanical integrators. The first accurate, completely electronic differential analyzer became available commercially in 1947 (2). Since that time many refinements have been made in devices of this type which have come to be called analog computers since the components may be connected to form an electrical analog of a physical system. While improvements have been made in accuracy of analog computers, the limit of accuracy obtainable at present is about 0.01 per cent and accuracy decreases as the number of analog computer components required in the problem increases.

The desire for greater accuracy and larger capacity than was possible with analog computers, along with a desire to retain the advantages of their component structure and ease of programming, led to the development of the digital differential analyzer (DDA). This machine performs much the same function as an analog computer but the mathematical operations are carried out digitally rather than continuously (3). The first practical DDA became available in 1950 (4). While these machines were capable of greater accuracy and

had considerably more capacity than most analog computers, they were considerably slower since the components were processed serially rather than simultaneously. In the early 1950's a great deal of work was expended on development of the DDA and many models became available commercially. The published literature on digital differential analyzers through 1956 has been collected and an extensive treatise published by Forbes (5).

Rapid advances in digital computers and in scientific programming languages (software) such as Fortran resulted in a lapse of general interest in the DDA from about 1956 until quite recently, although several important advances were made in DDA technology during this time. In 1958 a new type of DDA was announced in which all of the integrators coexisted physically rather than all components using one arithmetic unit (6, 7). A further development of a commercially available unit of this type was announced in 1962 (8). At present, high speed digital differential analyzers exist with digital computer accuracy although they are not widely used because of the investment required for such a special purpose machine.

In parallel with development of DDA's consisting of separate special purpose machines have been attempts to combine the advantages of DDA's and analog computers with those of general purpose digital computers. One such approach has been the development of hybrid computers in

which integration is performed continuously by electronic components while arithmetic and logic operations are performed digitally (9, 10, 11, 12). These systems are formed by physical connection of two different types of computers.

Another, more recent approach has been in the development of computer components in which digital and electronic methods are combined, the most significant part of the variables being represented digitally and the least significant part by an analog voltage (13, 14). No practical machine using this technique is yet available commercially.

A third approach to combining the advantages of analog computers and DDA's with the advantages of general purpose digital computers has been the development of hybrid programming systems in which analog or DDA components are simulated on a general purpose digital computer. This might be considered as software hybridization as opposed to the hardware hybridization approach described above. The advantage of systems of this type is that only one computer is required. With a well designed system of this type all of the advantages of serial digital differential analyzers could be obtained using only a general purpose digital computer without the additional investment for a special purpose DDA.

The basic digital computer routines required to simulate DDA components on a general purpose digital computer were developed by Selfridge (15). Several workable systems have been developed for utilizing these techniques. Lesh and

Curl (16, 17, 18) developed an interpretive system (DEPI) for the Datatron 204 computer which simulated the components of an analog computer. Slayton (19) developed a system (DIDAS) for the I.B.M. 704 which simulated the component structure of a DDA. Both systems used more accurate numerical integration procedures than the simple rectangular integration method of the conventional DDA (20). A fourth order Runge-Kutta method (21) was used with DEPI, while DIDAS employed a forward integration technique based on the formulas of Adam's method (22).

Recently, two more systems of this type have been developed. Both the system developed by Hurley and Skiles (23) (DYSAC) for the CDC 1604 computer, and the one developed by Gaskill et al. (24) (DAS) for the I.B.M. 7090, are related to DEPI in that they simulate analog computer components. Compiling methods are used with DAS while DYSAC is a semi-interpretive system. Both require special input languages. As with DEPI, DYSAC uses a fourth order Runge-Kutta integration method, while DAS uses a relatively inefficient rectangular integration technique.

While all of the above systems are programmed in terms of individual computer components, they differ in the types of components available, in the methods by which the component interconnections are coded and used as input to the digital programs, and in the efficiency with which this coding is

utilized by the digital program in executing the problem solution.

A system developed by Stein et al. (25, 26, 27) uses a somewhat different approach. It is programmed in terms of analog components as in the above systems but the input coding is used as input to a compiler which generates differential equations describing the system which are then solved by conventional digital methods. It utilizes the Fortran compiler as an intermediate in producing the object program. The types of problems which it can handle are more restricted than for the systems described above.

Ordinarily, the opposite of the operation which this system performs is what is desired. That is, the differential equations for the system are known while the analog components required and their interconnection are desired. Green et al. (28) describe a digital computer program which produces an analog computer program from the known differential equations. Palevsky and Howell (29) describe a generalized differential equation solving digital computer program by means of which the set of differential equations may be solved quite easily on a digital computer.

However, these latter methods have eliminated two of the major advantages of analog devices, i.e., the correspondence between parts of the physical system and parts of the coding diagram and the ability to add or remove easily a portion of the components to determine their effect on the

problem solution. These are the advantages of analog programming methods which are retained by the programming system developed in this work.

III. THE PROGRAMMING SYSTEM

A. Introduction

A system has been developed which makes it possible to program the I.B.M. 7074 in a manner very similar to analog computer programming methods. Advantages of analog computers and digital differential analyzers are combined with the advantages of a general purpose digital computer to yield a program which employs, as input, standard data cards prepared from a diagram analogous to an analog computer diagram. The capacity is much larger than that of most analog computers and the problem of scaling is eliminated. It may be used for more general boundary value problems than the initial value problems ordinarily solved on analog computers. Also, it is better suited for non-linear problems and control problems involving transport delay.

While the system may be considered to be essentially a simulation of a digital differential analyzer it possesses several important advantages over the conventional DDA. While it is related to the DDA and analog computer simulator systems described previously, the overall programming system is different from and superior to any of these systems.

DIAN makes it possible for anyone who has access to a large digital computer to, in effect, also have access to the equivalent of a large analog computer.

The system is compatible with Fortran and may be used as a part of a larger Fortran program. When used by itself DIAN does not require compilation, since it is in basic machine language.

B. Computer Components

In its present form, DIAN contains eight basic components. The method of representing them, the mathematical operations which they perform, and the relationships between inputs and outputs are shown in Figures 1, 2, 3, 4, and 5.

As with an analog computer or conventional DDA the most important component of DIAN is the integrator which is shown schematically in Figure 1. There are two types of inputs to the integrators, a single primary input and the secondary inputs of which there may be from one to three.

If the integrator were analogous to an analog computer integrator, the primary input would always be the differential of the independent variable. (This is the case with DYSAC, DAS, and DEPI.) However, the primary input to an integrator of DIAN may be the differential of any function of the independent and dependent variables. This provides an advantage over analog computers since it makes it possible to generate many non-linear functions by the use of combinations of the basic components. In this respect DIAN more closely resembles a DDA than an analog computer (30).

$$Y = \int_{t_0}^{t} (dy_1 + dy_2 + dy_3) + Y_0 \qquad \text{stored in integrator}$$

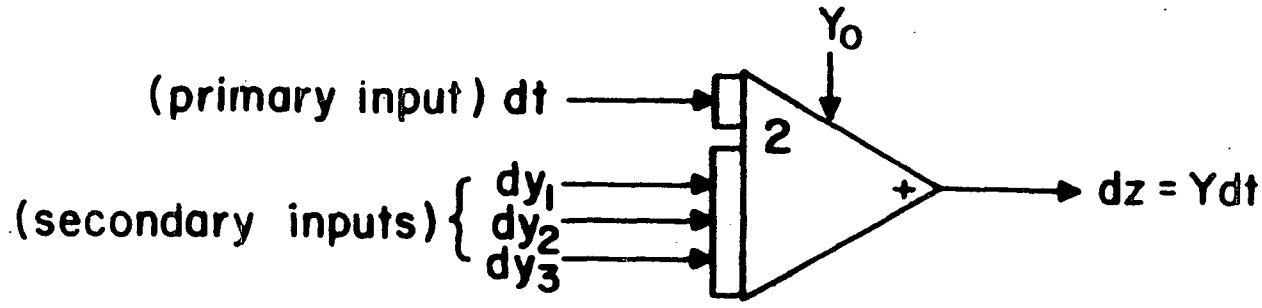
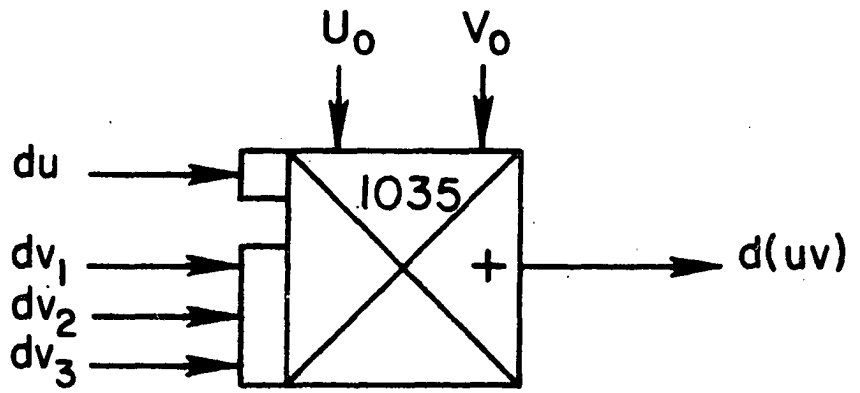


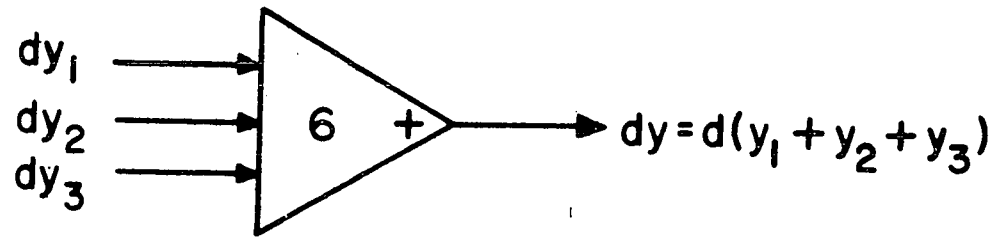
Figure 1. Integrator diagram



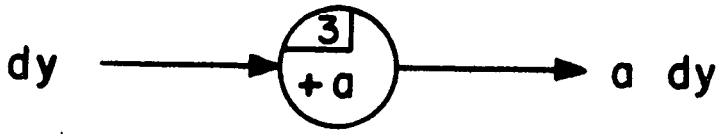
$$dv = dv_1 + dv_2 + dv_3$$

FUNCTION MULTIPLIER

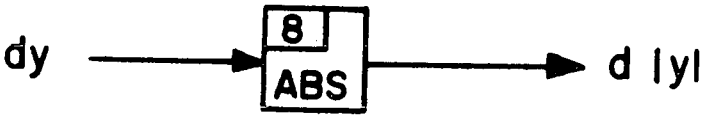
Figure 2. Diagram of a function multiplier



SUMMER

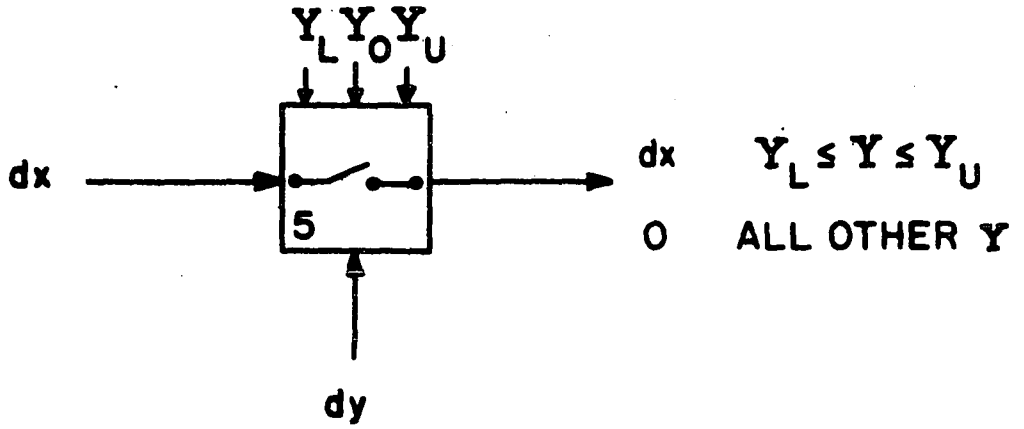


CONSTANT MULTIPLIER



ABSOLUTE VALUE GENERATOR

Figure 3. Diagrams of summer, constant multiplier, and absolute value generator



RELAY

TRANSPORT DELAY GENERATOR

Figure 4. Diagram of relay and transport delay generator

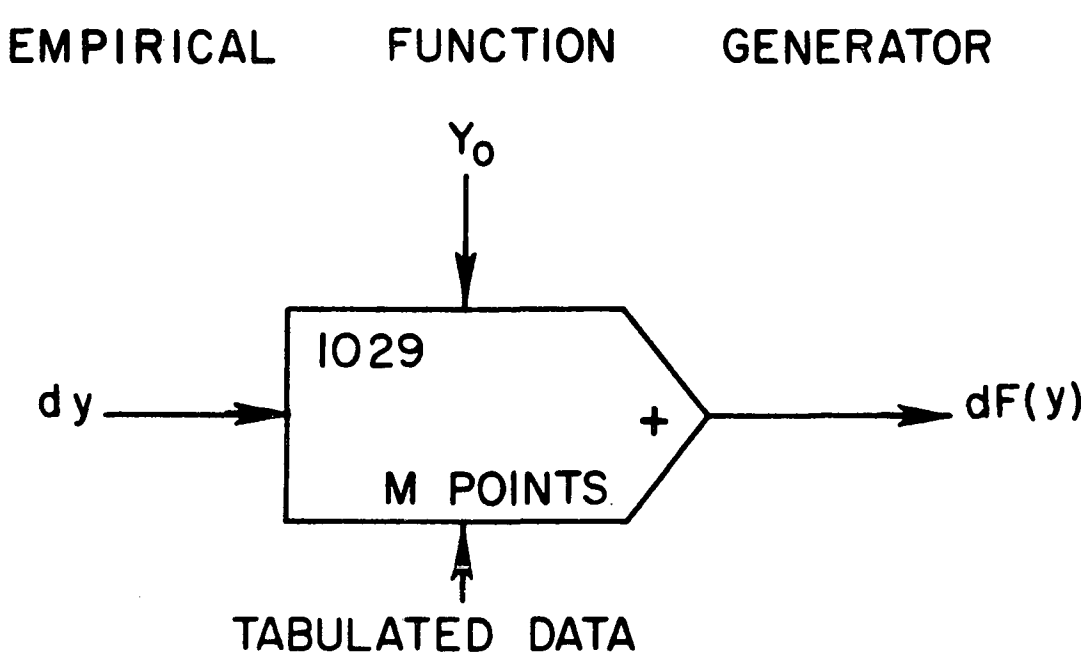


Figure 5. Empirical function generator diagram

Function multipliers, shown in Figure 2, are used to generate the differential of the product of two functions. As with the integrators, one of the functions may consist of the sum of several other functions. The operation of multiplication could be accomplished by the use of two of the DDA type integrators. However, if one considers the finite difference approximation,

$$U_{n+1} = U_n + (\Delta U)_n$$
$$V_{n+1} = V_n + (\Delta V)_n$$
$$(UV)_{n+1} = (UV)_n + U_n(\Delta V)_n + V_n(\Delta U)_n + (\Delta U \Delta V)_n,$$

the last term would be lost if integrators were used. The function multipliers are designed to include this term.

Three types of arithmetic components, the summer, constant multiplier, and absolute value generator, each of which requires only one type of input, are shown in Figure 3. Constant multipliers are analogous to analog computer potentiometers but the constant is not restricted to values less than one.

Transport delay generators, illustrated in Figure 4, allow an exact representation of time lag due to material transport. This feature can be very useful in control problems and is another advantage which the system has over analog computers since it is difficult to obtain satisfactory transport delay without distortion on an analog computer (31).

The relay, shown in Figure 4, may be used for branching during the problem solution by adding or dropping functions, depending on the value of a control variable.

Empirical function generators, which are represented schematically as shown in Figure 5, permit functional relationships to be defined empirically in terms of discrete data points.

The normal relationships between inputs and outputs of the components are as shown in these figures. However, the sign of the output of any component may be reversed if desired. This is indicated on the diagram by replacing the plus sign by a minus sign.

In addition to the above components, each problem requires one or more independent variable generators which supply initial and final values of the independent variable, the increment of the independent variable used in the numerical integration procedure, and the number of these increments per output interval. By the use of several independent variable generators the integration increment size or the output frequency may be changed at any predetermined point in the computation. These components have no input and their output is the differential of the independent variable.

C. Problem Preparation

For each problem, a computer diagram showing the components required and their interconnection is prepared in

much the same way as for an analog computer.

For example, to solve an n^{th} order ordinary differential equation, first solve for the highest order derivative, then multiply by the differential of the independent variable. That is, an equation in the form

$$F(t,y, \frac{dy}{dt}, \ldots\ldots, \frac{d^n y}{dt^n}) = 0$$

is rearranged to the form

$$d(\frac{d^{n-1}y}{dt^{n-1}}) = G(t,y, \frac{dy}{dt}, \ldots\ldots \frac{d^{n-1}y}{dt^{n-1}})\, dt$$

Next, assume the differential on the left is available as a secondary input to an integrator which uses dt as the primary input. The output of this integrator, $d(\frac{d^{n-2}y}{dt^{n-2}})$, may be used as a secondary input to another integrator with a primary input of dt to produce $d(\frac{d^{n-3}y}{dt^{n-3}})$. Continuing in sequence all the derivatives will be generated. The output of the last integrator will be ydt. The term tdt is generated by using dt as both the primary and the secondary input to an integrator. When all terms of the function Gdt have been generated by properly combining the various derivatives, they are combined to form $d(\frac{d^{n-1}y}{dt^{n-1}})$ which is the input for the first integrator.

If special functions are required they may be generated from combinations of basic components. Examples of the method

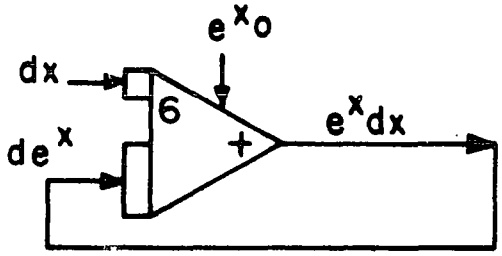
are shown in Figures 6 and 7. Many different non-linear functions of the independent and dependent variables may be generated in this manner.

In preparing the flow diagram the integrator inputs need not be scaled since the digital computer uses floating point arithmetic.

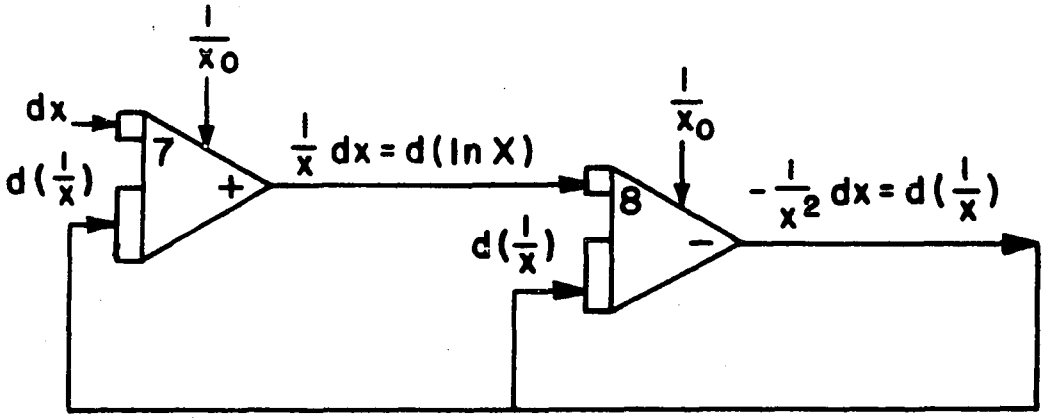
During preparation of the diagram each component is assigned a serial number from 1 to 1400 (500 with the 10K system). Integrators must have serial numbers between 1 and 1000 (350 with the 10K system), and function multipliers and empirical function generators must have serial numbers between 1001 and 1400 (351 and 500 with the 10K system). Independent variable generators must have a serial number of 0, regardless of how many of them are used for the problem. The total number of components possible is 1400 (500 with the 10K system). The number of empirical function generators plus transport delay generators is limited to 15 (10 with the 10K system).

Figures 8 and 9 illustrate a computer diagram for the solution of two simultaneous non-linear differential equations. Generation of the non-linear differential function $Ce^{-E/RT}dt$ is illustrated by Figure 8. In Figure 9 this non-linear expression is incorporated into the complete diagram for solving the equations:

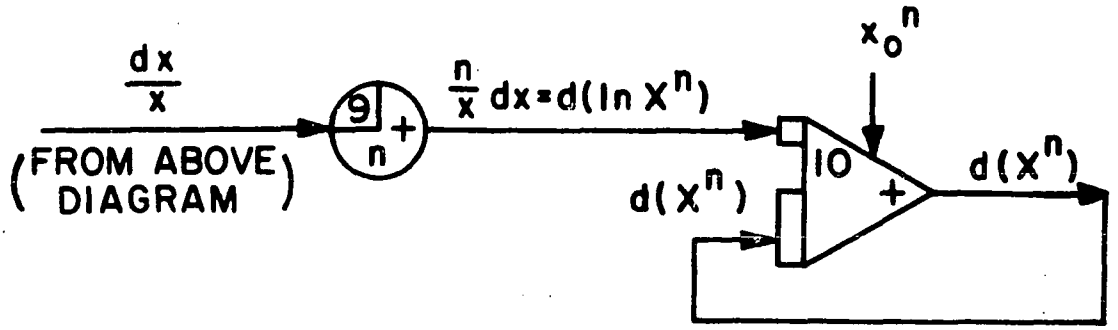
$$dC = a_1dt - b_1Cdt - K_1Ce^{-E/RT}dt$$



EXPONENTIAL FUNCTION

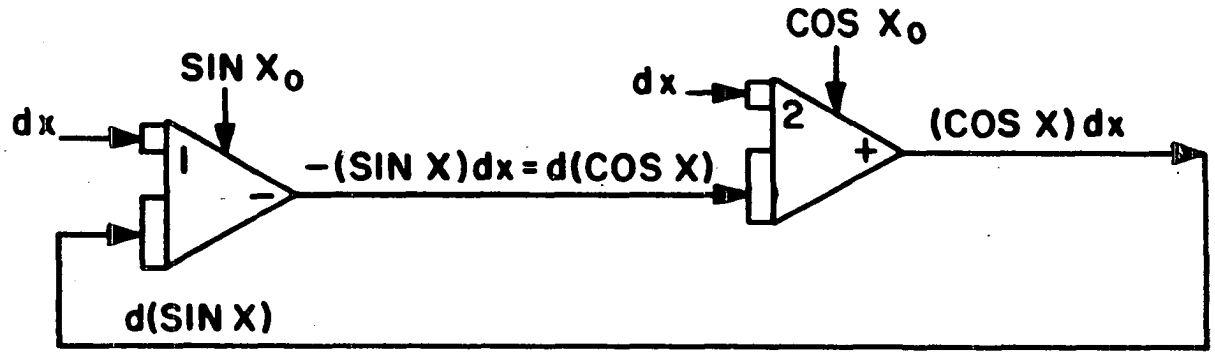


LOGARITHMIC AND RECIPROCAL FUNCTIONS

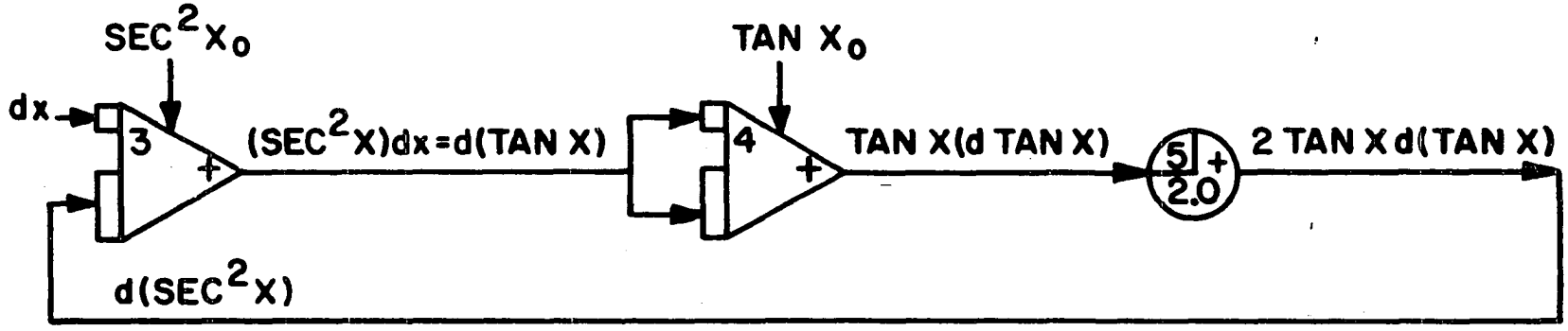


VARIABLE TO A POWER n

Figure 6. Generation of non-linear functions



SINE – COSINE GENERATOR



TANGENT GENERATOR

Figure 7. Generation of trigonometric functions

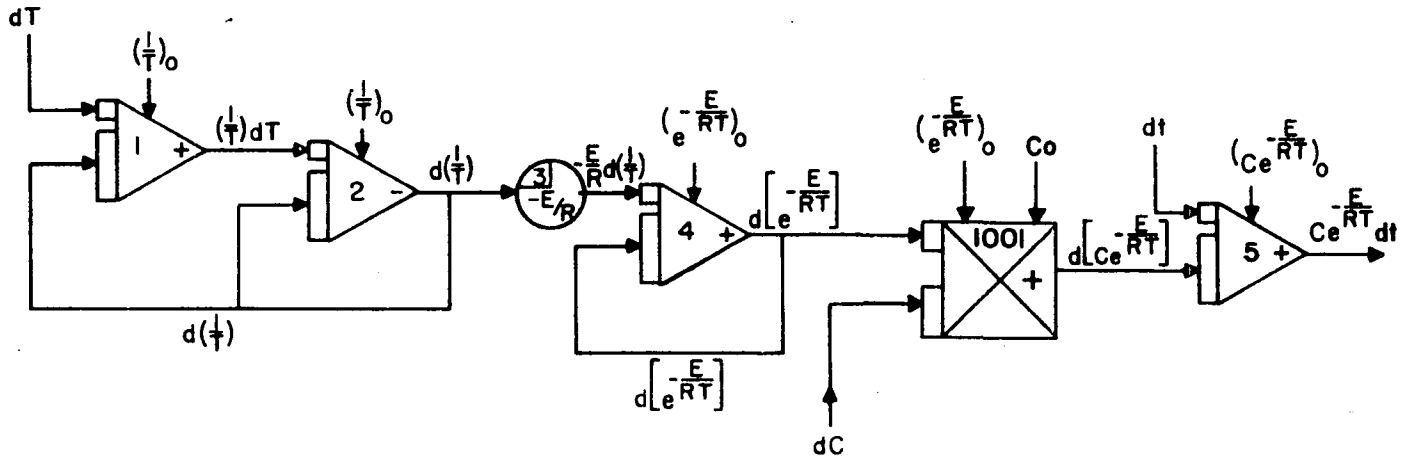


Figure 8. Generation of the function $Ce^{-E/RT}dt$

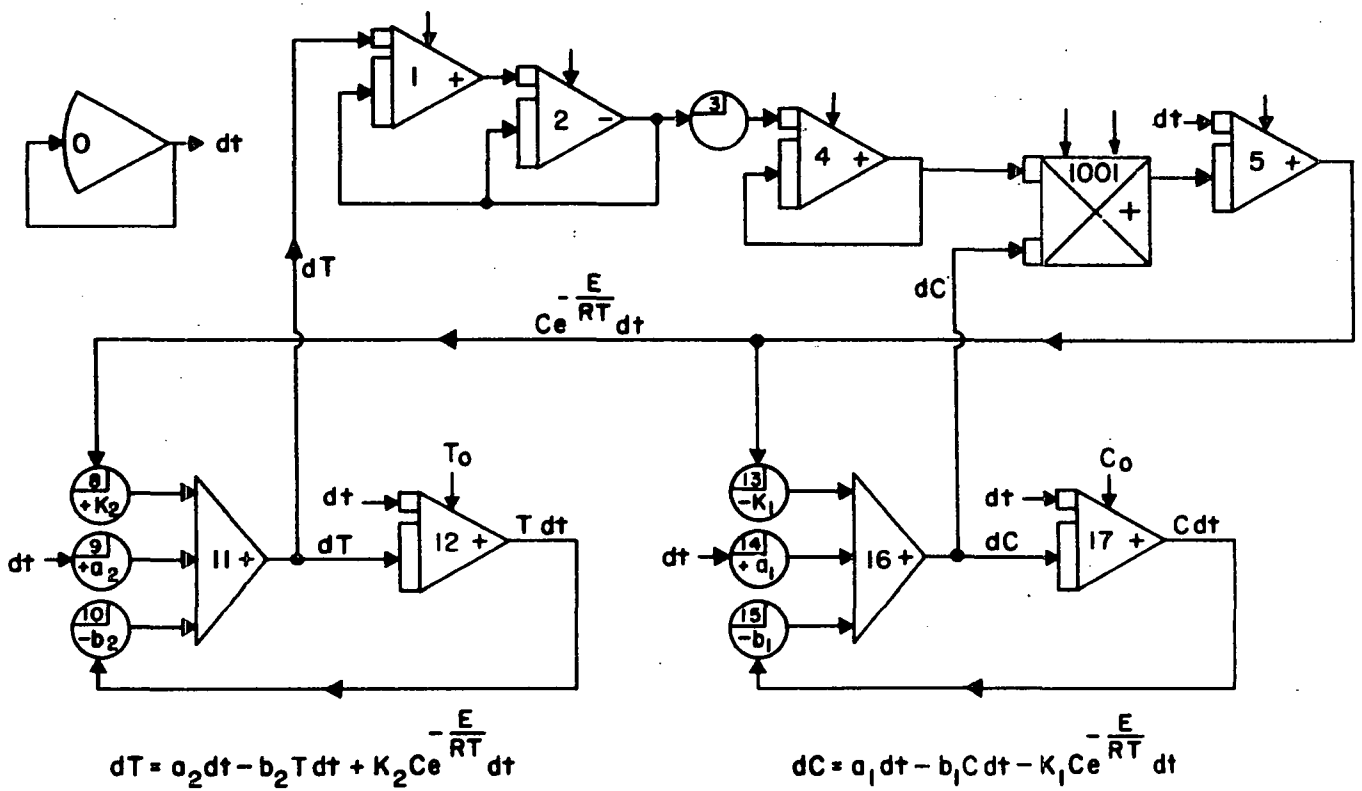


Figure 9. Diagram for solution of two non-linear differential equations

and

$$dT = a_2dt - b_2Tdt + K_2Ce^{-E/RT}dt$$

D. Input Coding

1. Component cards

For input to the digital computer each of the components is represented by one data card containing (a) the component serial number, (b) a code number identifying the type of component, (c) primary and secondary input specifications consisting of the serial numbers of other components whose output form the input to the component, plus (d) additional information such as initial values for integrators, constants for constant multipliers, delay times for transport delay generators, etc.

The detailed coding of the input cards for the various types of components are shown in Figures 10-17. Integrator cards contain the component number, identification code, initial value, one primary input, and three secondary inputs and are read by the digital computer according to the Fortran statement - FORMAT (2I5, E15.7, 4I5).

Function multipliers use a similar format except that they require two initial values for Uo and Vo in that order, where U is the primary input variable and V is the sum of the secondary input variables. They are read according to the

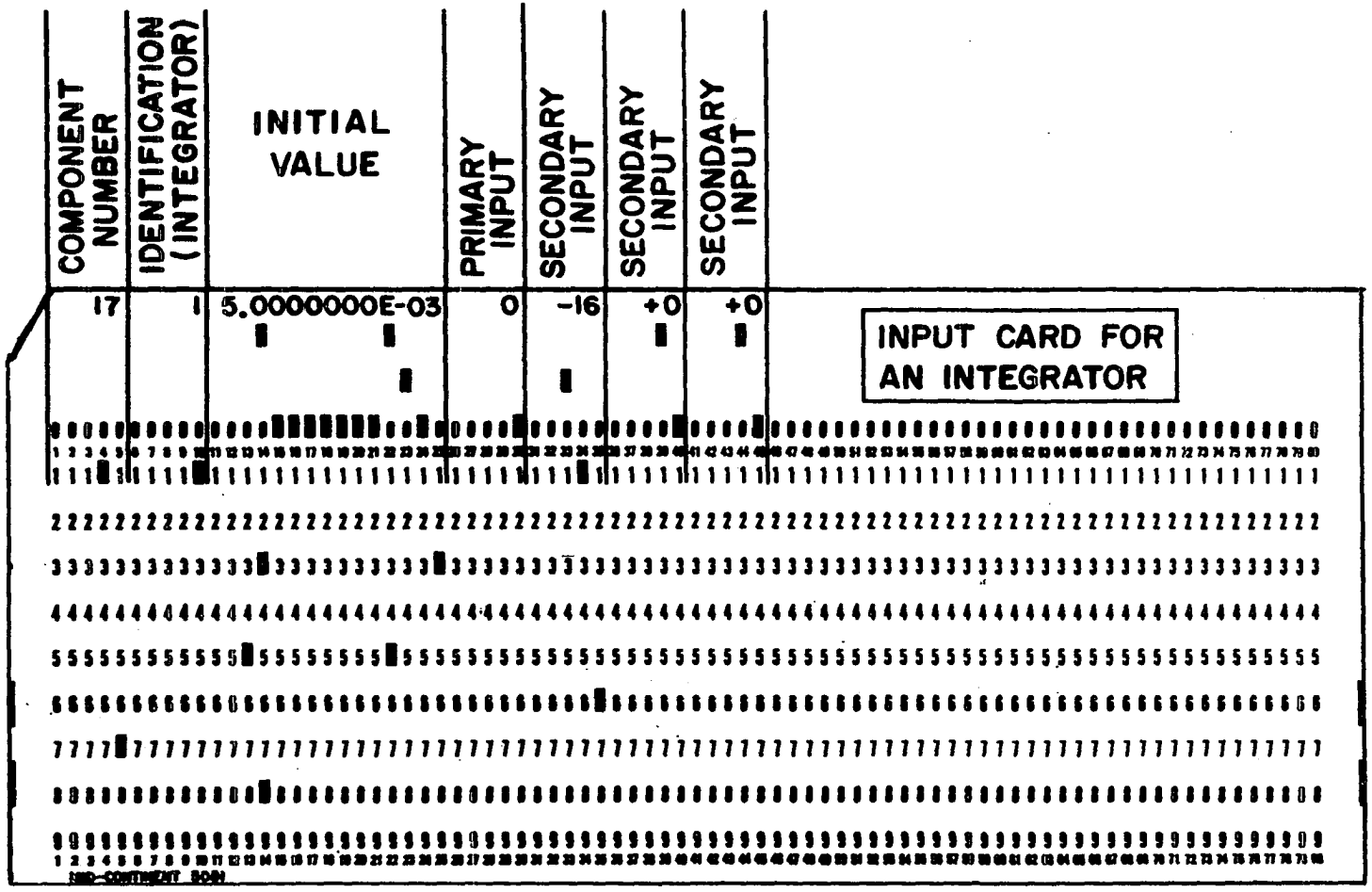


Figure 10. Input card for an integrator

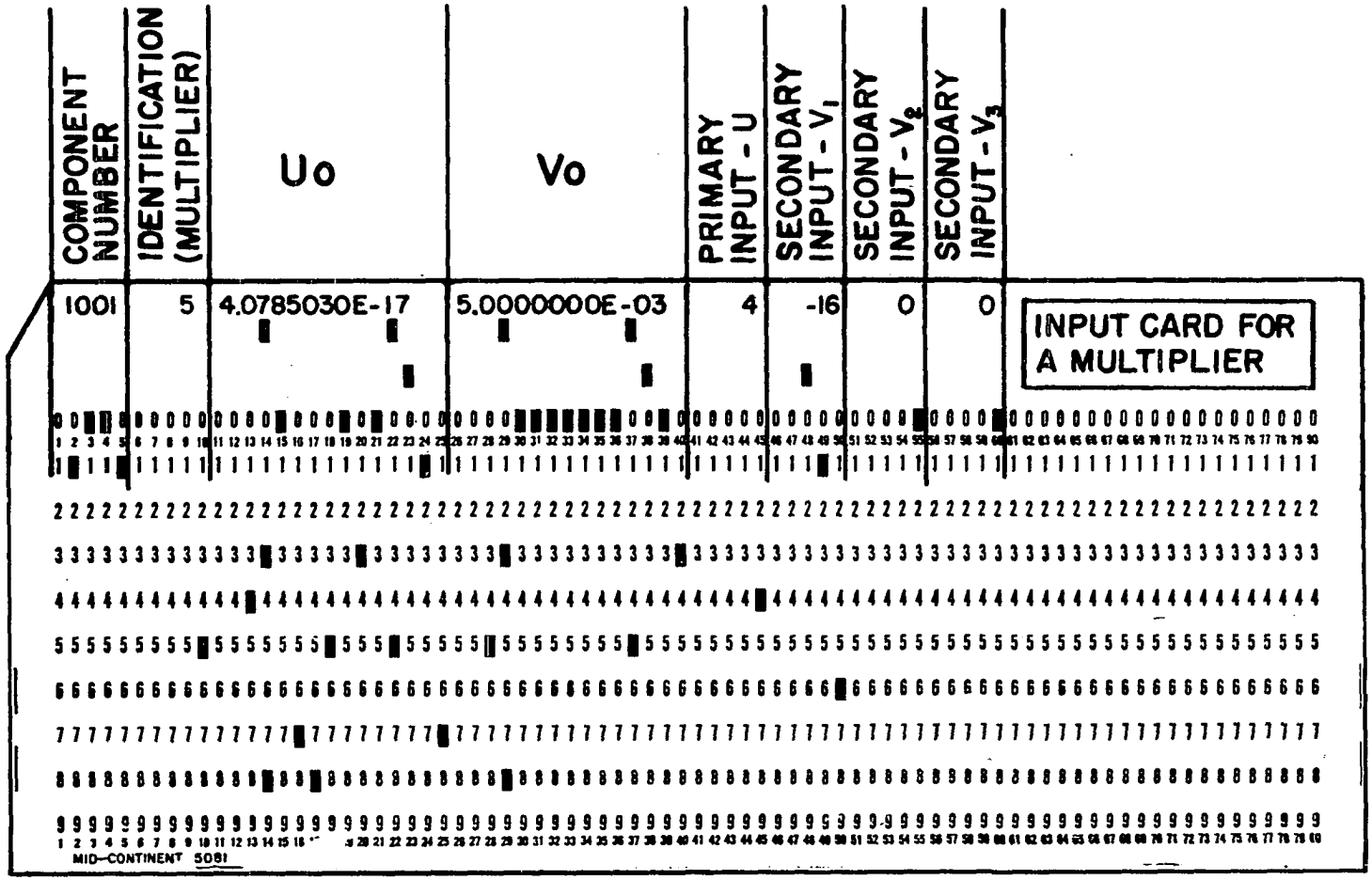


Figure 11. Input card for a function multiplier

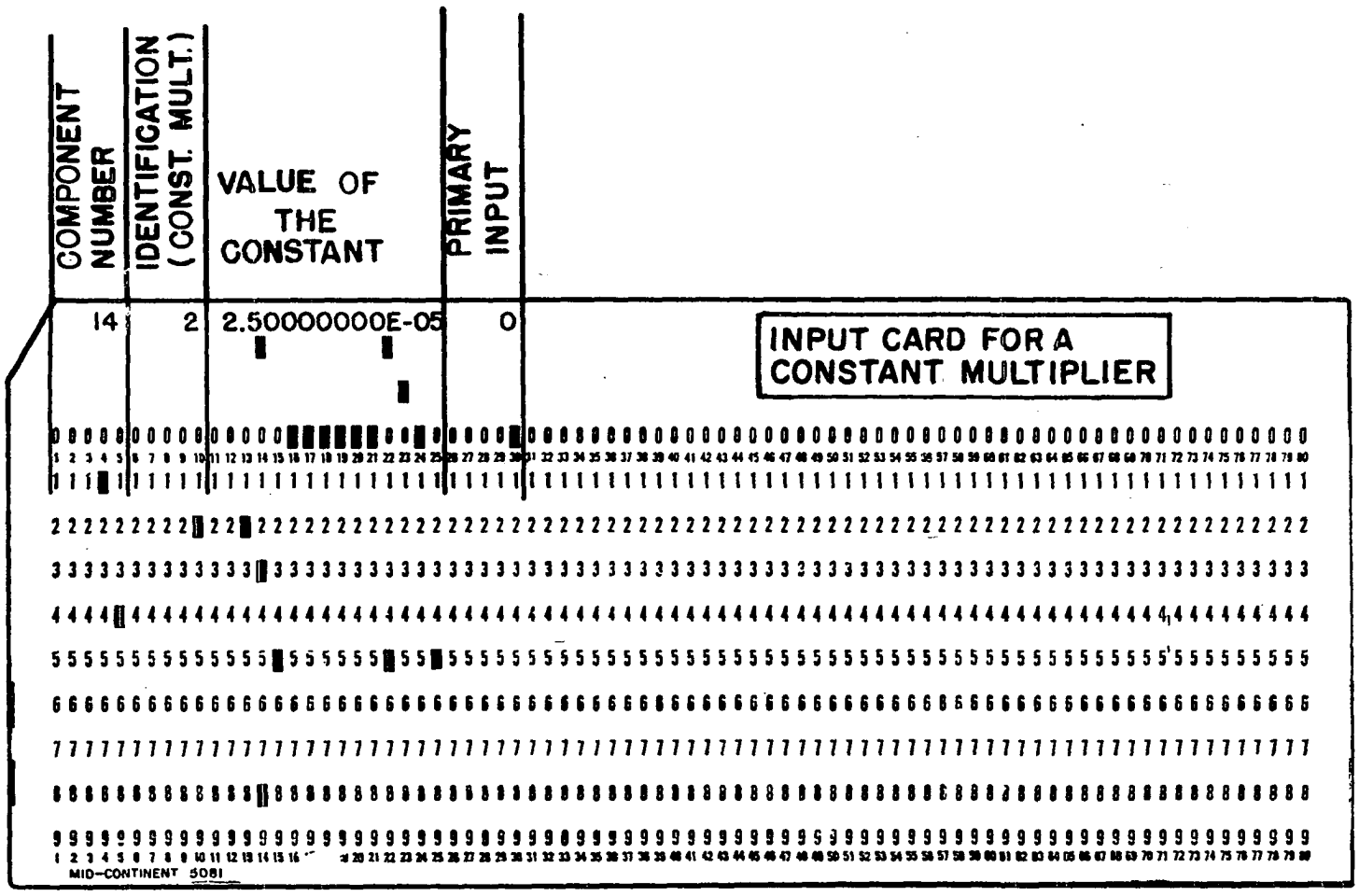


Figure 12. Input card for a constant multiplier

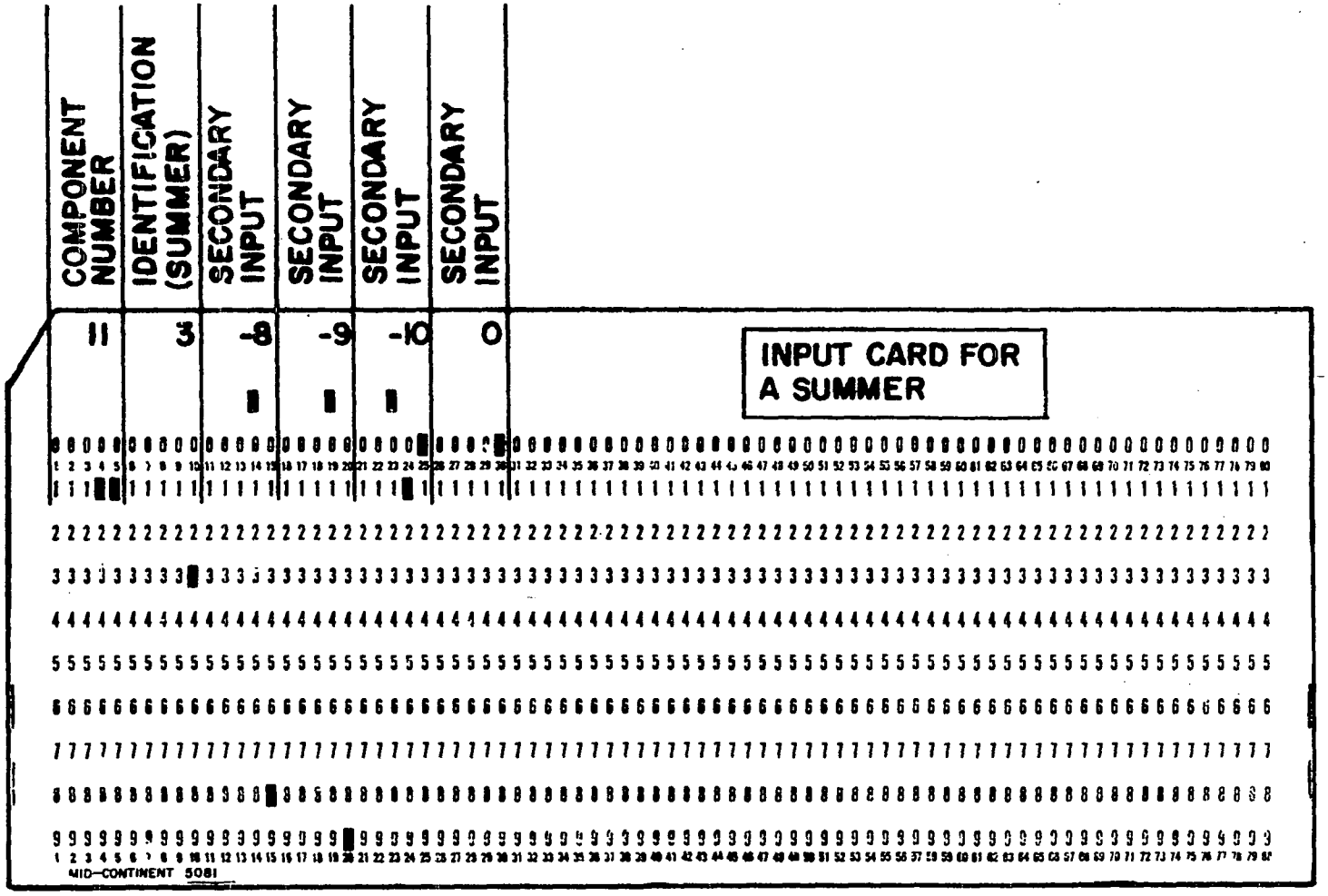


Figure 13. Input card for a summer

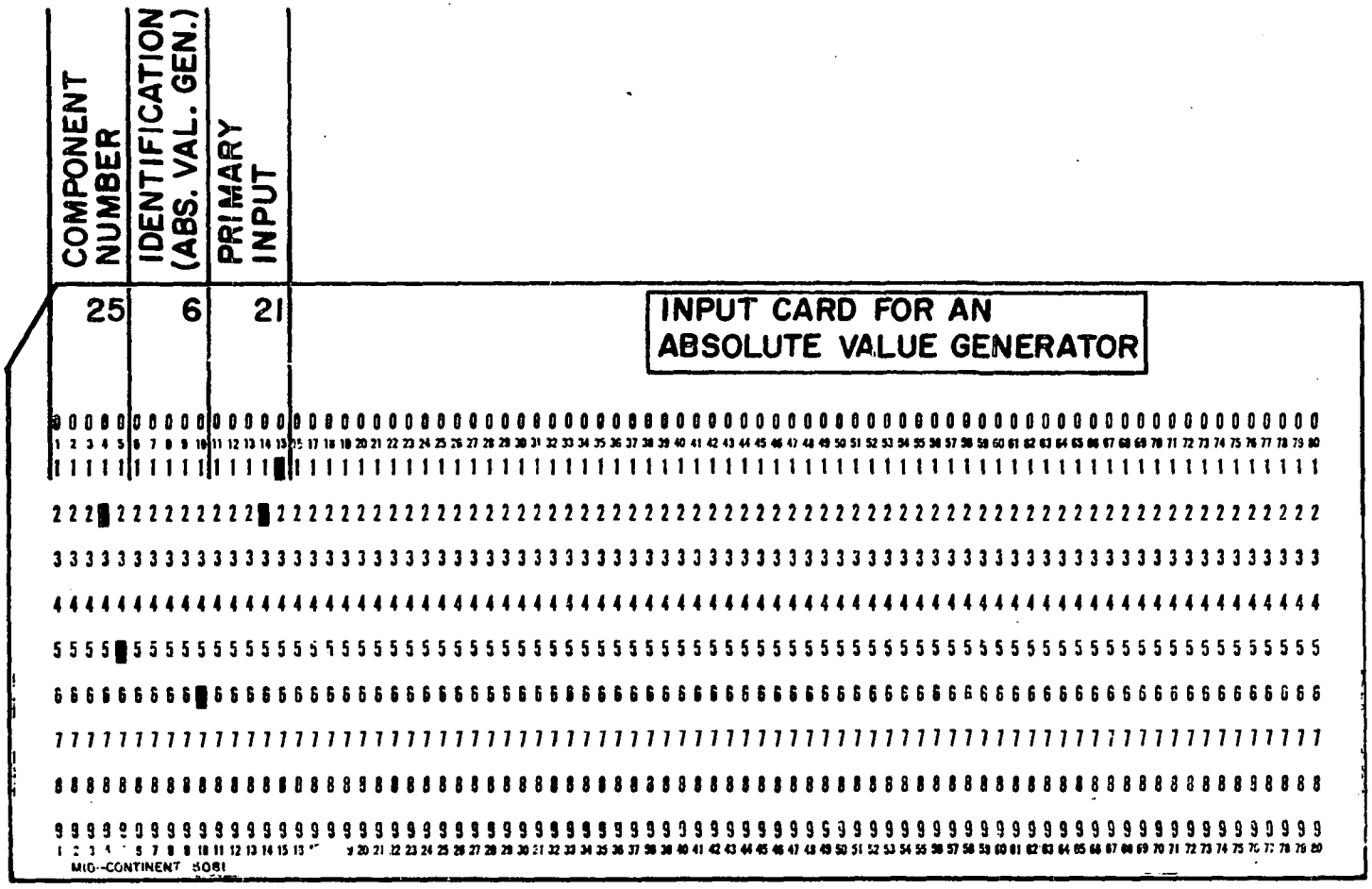


Figure 14. Input card for an absolute value generator

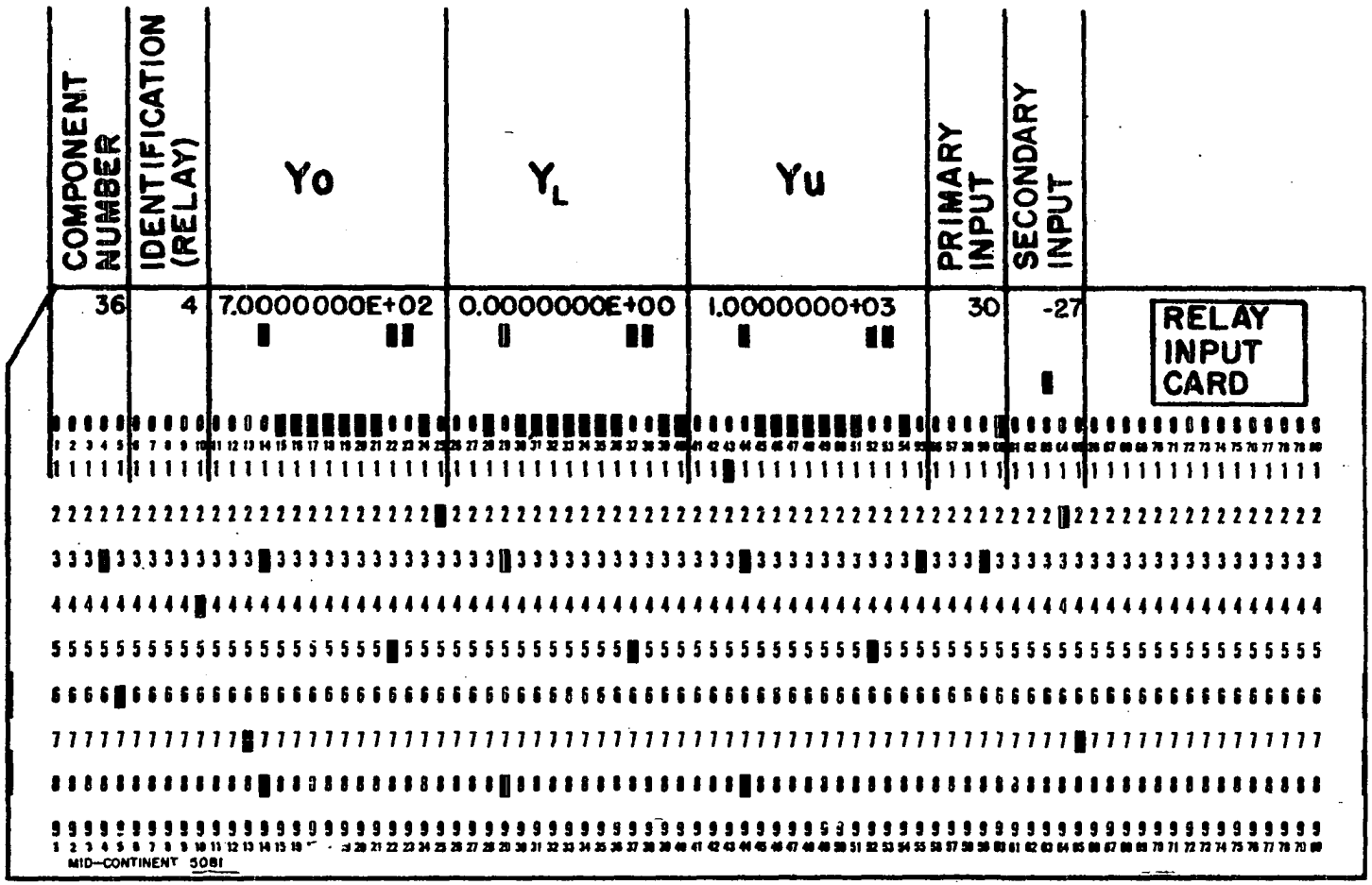


Figure 15. Input card for a relay

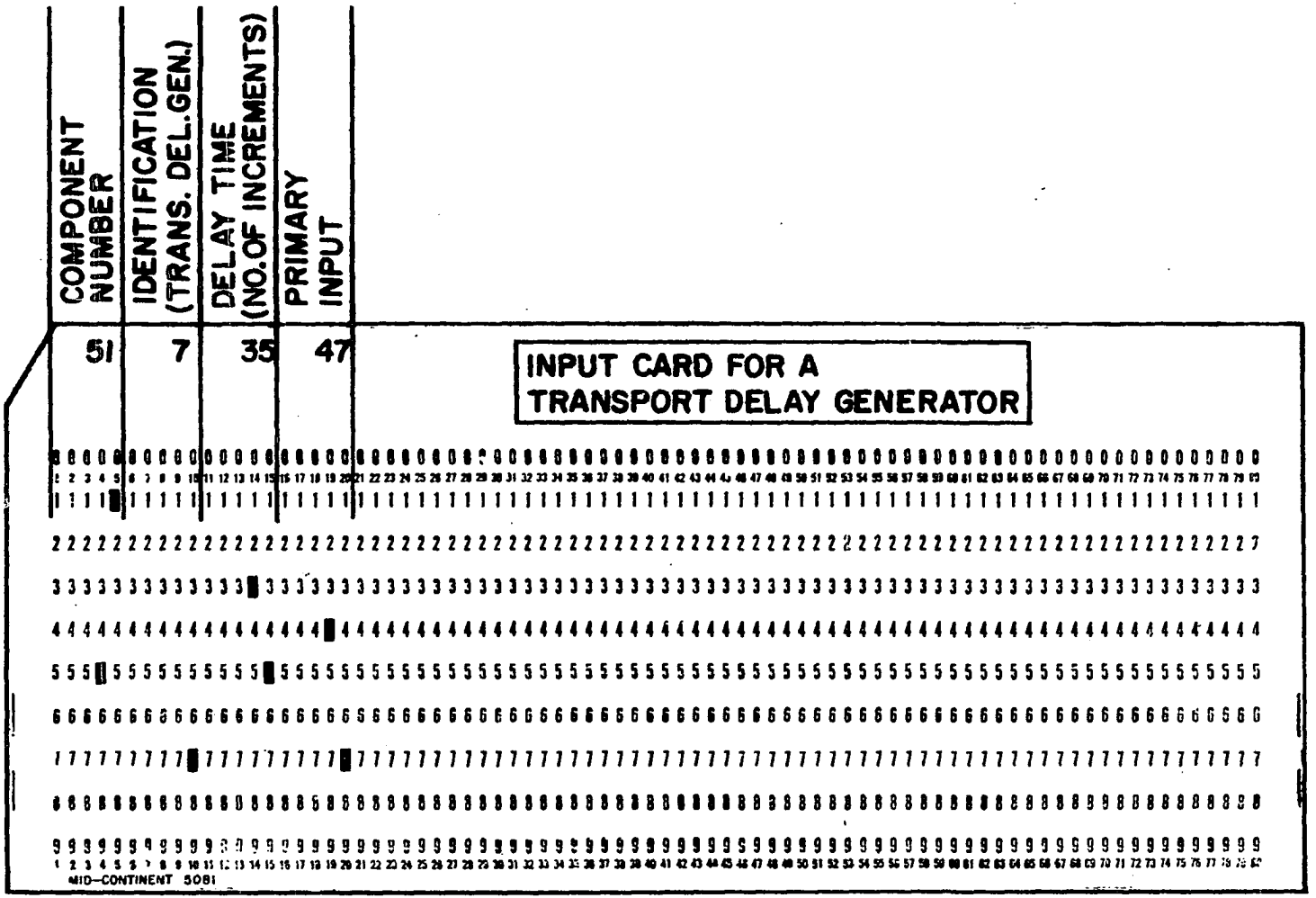


Figure 16. Input card for a transport delay generator

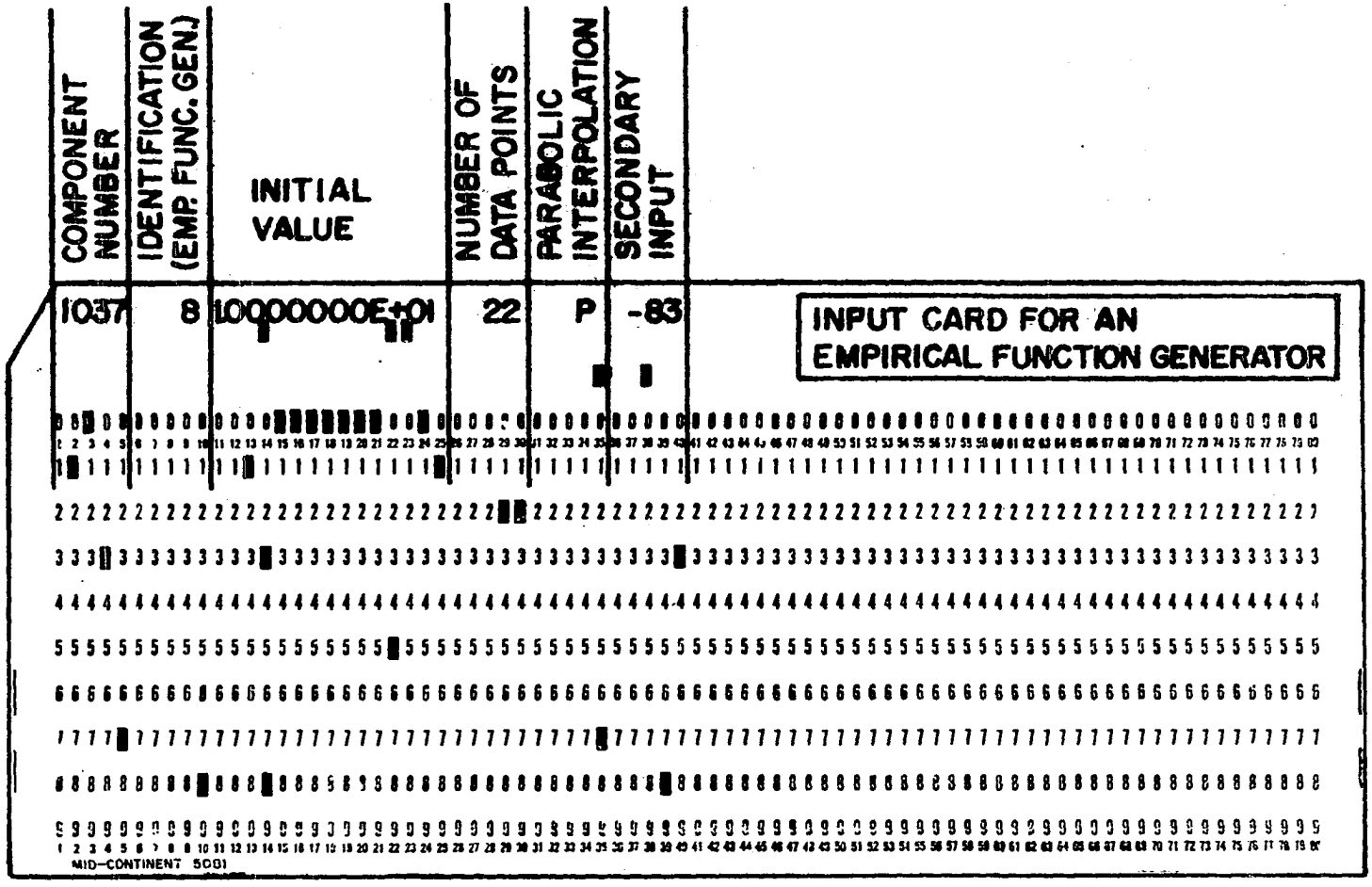


Figure 17. Input card for an empirical function generator

Format statement - FORMAT (2I5, 2E15.7, 4I5).

Input cards for constant multipliers contain the component number, identification code, the constant, and one primary input. For this component the computer uses the Format statement - FORMAT (2I5, E15.7, I5).

Input cards for summers contain component and identification numbers plus four secondary inputs and are read in the form - FORMAT (6I5).

Cards for absolute value generators contain only the component and identification numbers plus one primary input and are read in the form - FORMAT (3I5).

Transport delay generator cards require component and identification numbers, the number of independent variable increments in the delay time, and one primary input. They use the statement - FORMAT (4I5). Up to fifteen transport delay generators may be used for a single problem and the length of the time delay may be up to 100 increments of the independent variable. The increment size may not be changed during the solution of problems involving transport delay generators.

Input cards for relays contain component and identification numbers, the initial value of the control variable, lower and upper limits on the range of the control variable, plus one primary and one secondary input specification in that order. The secondary input variable is the control variable (for example, dy in Figure 4). These cards are read

by the computer according to the fortran statement - FORMAT (2I5, 3E15.7, 2I5).

Cards for empirical function generators contain the component number and identification code, the initial value of the input variable, the number of data points to be used with the component, the letters P or L to specify linear or parabolic interpolation between points, and one secondary input. These cards are read in the form - FORMAT (2I5, E15.7, I5, A5, I5). Each empirical function generator card is followed by cards containing the data points in order of increasing value of the input variable, each data point consisting of two numbers, a value of the input variable and a corresponding value of the output variable, according to the specification - FORMAT (4E15.7).

The identification codes for the various types of components are integers one through eight as listed in Table 1.

Table 1. Component identification codes

COMPONENT	CODE
Integrator	1
Constant Multiplier	2
Summer	3
Relay	4
Function Multiplier	5
Absolute Value Generator	6
Transport Delay Generator	7
Empirical Function Generator	8

In specifying inputs for components, primary inputs are always positive and secondary inputs are negative except in the following case. Components which may have more than one secondary input require dummy secondary inputs if less than the maximum number are used. Positive zeros may be used for this purpose and they will be ignored by the DIAN system. For example, in Figure 10, the integrator which this card represents required only one secondary input and, consequently, the input card must contain two dummy inputs.

The independent variable generator is a special type of component which has no input and whose output is the differential of the independent variable. Coding cards for these components contain a zero in the first field as an identification code and component number. Initial and final values of the independent variable (the integration limits) and the independent variable increment are specified by the next three numbers. The last number specifies the frequency with which output is produced by the program by specifying the number of independent variable increments in each output interval. Independent variable generator cards are prepared to fit the format statement - FORMAT (I5, 3E15.7, I5). Up to thirty of these components may be used with a single problem in order to change increment size or output frequency during the computation. This corresponds to breaking up an integral into several parts.

2. Output specifications

Output for a problem is specified by listing the serial numbers of the integrators which hold the integrated values of the desired output variables and by supplying a format for printing the output line. At any time during the computation, a particular integrator holds the current value of its secondary input variable. The independent variable is always an output variable and need not be specified. A maximum of forty-seven other output variables may be specified on one to eight input cards, each of which contains six integrator numbers and which are read according to a FORMAT (6I5).

These output variables are printed in the order in which they are listed on the cards (except that the independent variable is always the first variable printed) according to a Fortran "FORMAT" statement supplied as input with the problem. The output format card contains a seventy character Fortran "FORMAT" statement excluding the word "FORMAT". That is, a left parenthesis is placed in column one and a right parenthesis in column seventy. Any blanks in the statement are ignored so the statement need not completely fill the remaining sixty-eight columns.

3. The complete input deck

Since several problems may be run sequentially in a multifile run, the first input card is a control card

containing the number of problems in the run in columns one through five. This number is right justified in the number field. This is followed by the individual coding decks for each problem.

The first card of a problem input deck is a title card containing a problem title of up to seventy characters which is reproduced on the first page of output. A "B" as the first character of this title designates a boundary value problem (see the following section). Any other character may be used in position one of this card for initial value problems. The second card is a control card which supplies information needed to properly read the following cards. It contains the total numbers of the various types of components, the number of independent variable generators, and the number of output variables (excluding the independent variable), in the order: integrators, constant multipliers, summers, relays, function multipliers, absolute value generators, transport delay generators, empirical function generators, independent variable generators, and output variables. This card is read according to a FORMAT (10I5) statement.

Following these two cards are the input cards for all of the components. They are grouped according to component type. That is, all of the integrators are grouped together, then all of the constant multipliers, etc., in the order of the identification codes as listed in Table 1. Data cards for empirical function generators are placed immediately

following the card for the component with which they are used.

Cards for independent variable generators are included next, followed by the output variable specification cards and the output format card.

If more than one problem is included in the run the next card is the title card for the next problem.

This entire input deck is used as input data for the object programs compiled from the DIAN source programs.

E. Application of the System to Boundary Value Problems

DIAN contains provisions for a subroutine which can be used to extend its applicability to more general types of boundary value problems requiring an interpolation procedure and iterative solution. For such problems a final rather than initial value of one of the variables is known. An additional input card is required to provide data necessary to carry out the iterative solution.

Since the particular components whose initial values are changed after each iteration depend on the nature of the problem it is not possible to write a single subroutine which would be applicable to all boundary value problems. Therefore, programming of boundary value problems requires rewriting and recompilation of the Fortran subroutine. A sample subroutine for a particular boundary value problem is included as a part of the library system.

In order to avoid recompilation of the main program for each boundary value problem the main program uses a subroutine called "INTERP" for every problem. When the system is being used for initial value problems a dummy subroutine is used which, if written in Fortran, would consist of the three statements:

```
SUBROUTINE INTERP
RETURN
END
```

The object deck for this dummy subroutine consists of a title card, one condensed card, and one execute card which can be prepared from examples in a Fortran reference manual without actually compiling the three statements.

For a boundary value problem this dummy subroutine should be replaced by the actual interpolation subroutine for that problem. The total core storage area which may be used for this subroutine is limited to 1100 words (600 with the 10K system).

F. Error Messages

Certain error conditions will cause the problem to be terminated automatically and a message typed on the console log sheet.

After each floating point arithmetic operation a check is made to determine whether the operation has resulted in

an attempt to generate a number larger than 10^{51}. If so, an error condition results and the message "Floating Overflow" is typed. This may result from an error in the input coding or from instability of the integration procedure caused by the use of too large an increment size.

When empirical function generators are used an error condition results if the value of the input variable is outside the range of the tabulated data. The message "Variable Exceeds Range of Table" will be typed. This may also result from integration instability.

If a read or write operation on one of the two work tapes results in ten consecutive error conditions, the message "Ten Tape Errors" will be typed.

IV. THE SOURCE PROGRAMS

A. Program Logic

There are three parts of the source program for the DIAN system. Input and output is handled by a program written in Fortran which serves as the control program. It contains two subprograms, one written in Fortran which is used in conjunction with boundary value problems and one written in Autocoder which actually performs the computations and comprises the main part of the system. These programs need be compiled only once to produce an object deck in machine language which uses the input decks described previously.

When the input coding cards are read by the input portion of the program, the numbers are stored sequentially in "Common" storage. An initialization procedure in the subroutine converts this block of storage into a sequence of machine language instructions which, in effect, establishes the component interconnections and provides linkages with closed subroutines which perform the mathematical operations. This initialization requires about forty microseconds per component and is executed only once for each problem.

The computational subroutines are entered by means of a block of branch points arranged as shown in Table 2. Each component has two entrance points, one for a normal computation and another when a reversal of the sign of the output is desired. For example, instruction number 8 in Table 2

Table 2. Entrance to computational subroutines

Instruction number		Autocoder instruction	
1		B	EFGM
2		B	DELAYM
3		B	ABSM
4		B	MULM
5		B	DSNM
6		B	ADDM
7		B	CONM
8		B	INTM
9	AA	B	BRANCH-1
10		B	INTP
11		B	CONP
12		B	ADDP
13		B	DSNP
14		B	MULP
15		B	ABSP
16		B	DELAYP
17		B	EFGP

would result in a branch to the integrator subroutine in such a way as to cause a reversal of the sign of the output of the component. Instruction 10 would cause a branch to the same subroutine for a normal computation. Note the symmetrical arrangement of the branch points about instruction number 9.

The initialization routine uses two constants in its computations. They are:

XZACOMD +4600210000

and

BLXCOMD +0200240000

The first operation performed by the initialization routine consists of determining the core storage address of instruction AA (instruction 9 in Table 2) and replacing the last four digits of BLXCOMD by this address. It then proceeds through the block of input data and adds XZACOMD to the contents of the first data word for each component and BLXCOMD to the second word. For each component, the first data word contained the component serial number and the second contained the component identification code. When the above constants are added to these words the result is that the first word of a data block for a component becomes a machine language instruction to load index number 21 with the component serial number and the second word becomes an instruction to load index number 24 with its address and branch to one of the instructions of Table 2, depending on the value of the identification code.

For example, an identification code of 1 would cause a branch to the address AA+1 (instruction 10 of Table 2) while a minus 1 would cause a branch to AA-1 (instruction 8 of Table 2).

The initialization routine also replaces the contents of the two core storage locations immediately following the end of the input data block by XZACOMD and BLXCOMD. When control is transferred to these instructions they cause index 21 to be loaded with a zero and control to branch to AA (instruction 9 of Table 2). This causes a branch to a logic

area of the DDA subprogram rather than to a computational subroutine.

A major difference between analog computers and conventional digital differential analyzers or DDA simulators such as DIAN is that all mathematical operations are performed simultaneously on an analog computer (parallel operation) whereas the digital method requires that the components be serviced serially.

In performing the computations using DIAN the digital computer works sequentially through the coding block assembled from the input data. For each component in the sequence the first instruction causes a working index to be loaded with the component serial number. The second instruction causes a branch to the appropriate closed subroutine. The remaining storage words for that component are values of indices to be used by the subroutine in determining the locations in the working storage blocks from which to obtain the necessary data for carrying out the mathematical operation. After the computation required to update the component has been completed, the results are stored in the appropriate registers and control is transferred to the instructions for the next component in the sequence.

After all of the components have been updated a check is made to determine whether the integration limit or an output point has been reached. If not, the independent variable is incremented and the servicing procedure repeated.

Five blocks of working storage, each containing one storage word for each component, are used in carrying out the computations. The five storage words associated with each component contain the necessary data to describe the status of the component at any time during the computation and to carry out the mathematical operation assigned to the component. These five registers are a Y register which contains the current integrated value of the input variable, an R register used in double precision accumulation of the inputs to a component, a DZ register which contains the output of the component, and YNM1 and YNM2 registers which hold the previous two values of Y for use with the numerical integration formulas. Individual components are distinguished by an indexing system which specifies the locations within the storage blocks of the words associated with the component. The value of the index used for this purpose is the serial number assigned to each component during coding.

B. The Integration Method

The numerical integration procedure used by DIAN consists of the following forward integration formulas:

1. $y_1 = y_o + (\Delta X)(y'_o)$

2. $y_2 = y_1 + (\Delta X)(\frac{3y'_1}{2} - \frac{y'_o}{2})$

3. $y_{n+1} = y_n + (\Delta X)(\frac{23y'_n}{12} - \frac{4}{3}y'_{n-1} + \frac{5}{12}y'_{n-2})$

These formulas may be derived by integrating Newton's backward interpolation formula and truncating before the first, second, and third differences respectively (32). Equation 3 is analogous to the Gregory-Newton formula in terms of forward differences (33). Similarly, Equation 2 is analogous to the trapezoidal rule, and Equation 1 is Euler's formula. The first two equations are only used for the first two increments in starting the integration procedure. To compensate for the resulting reduced accuracy, the integration increment size is automatically reduced temporarily at the beginning of the problem and after a change has been made in the increment size during the computation.

In an attempt to avoid this somewhat inefficient starting procedure, a version of the system using a fourth order Runge-Kutta integration method was developed. However, it was found that this technique was no more accurate than the above method for equal increment sizes while it required more computing time and considerably more working storage area. These disadvantages outweighed the ease of changing increment size with the Runge-Kutta procedure so the method described above was retained. Actually, unless increment sizes are changed very often, the inefficiency of the starting procedure does not significantly affect the total computer time required for solving a problem since a large percentage of

the computer time is used in input-output operations involving magnetic tape units.

C. Flow Charts

The operation of the three main sections of DIAN are represented in the following flow charts. Figure 18 represents the Fortran program which processes input and output for the system. The Autocoder subprogram (DDA) is represented in four parts in Figures 19-22. Figure 23 is a flow chart for the Fortran subprogram INTERP which is used with boundary value problems. The last three flow charts, Figures 24-26 represent three of the closed subroutines which are included within the DDA subprogram for carrying out the mathematical operations. These flow charts are for the subroutines used by integrators, transport delay generators, and empirical function generators. Since the subroutines for the other components are relatively simple, flow charts are not given for them.

Each of the flow charts is described in detail in the following pages. Complete listings for source programs are given in the Appendix.

1. Flow chart A, Figure 18; Fortran main program

Block A1: Problem is initialized, control variables read in, and problem title transferred to output.

Block A2: All component cards are read in and stored in

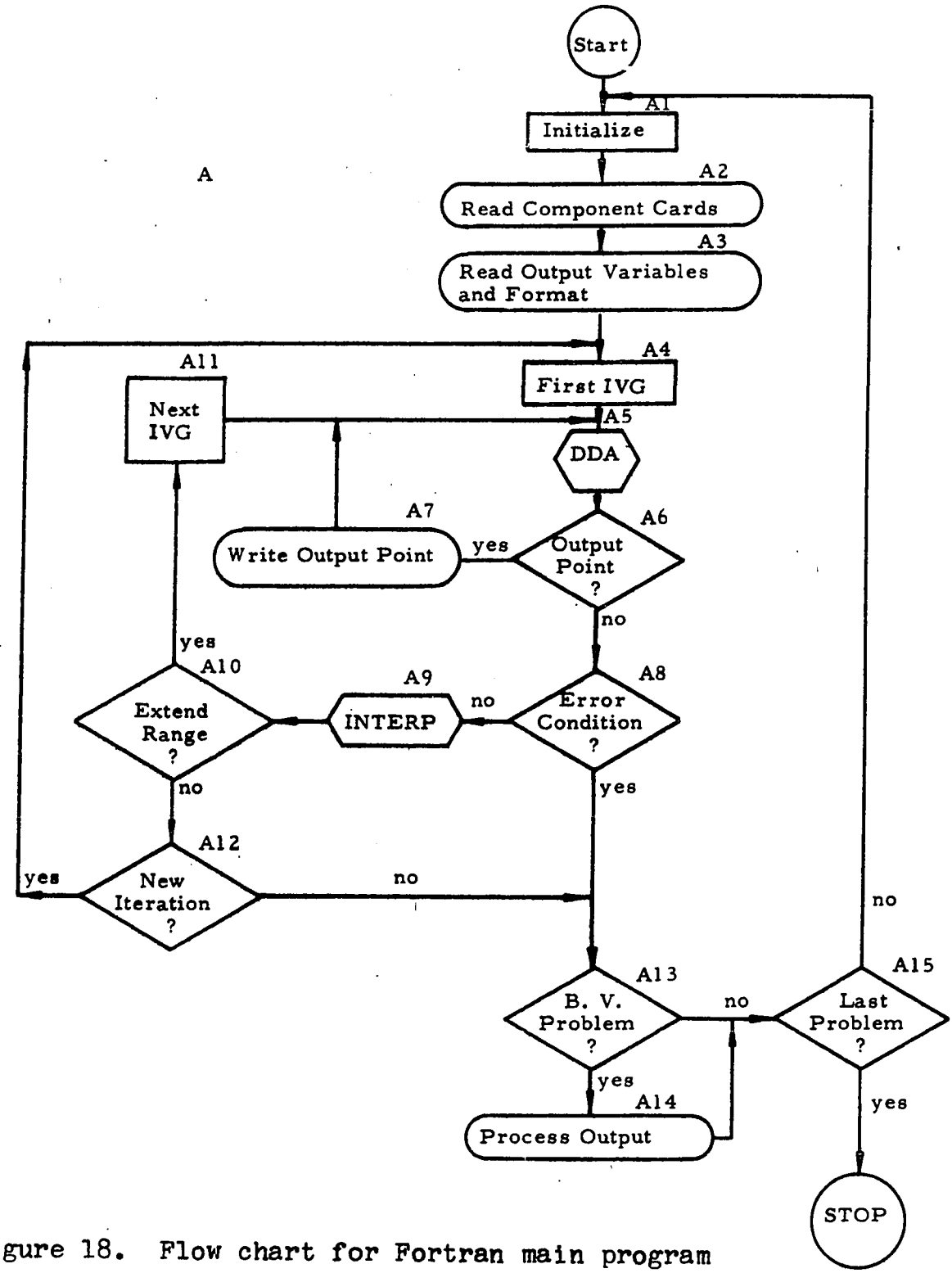


Figure 18. Flow chart for Fortran main program

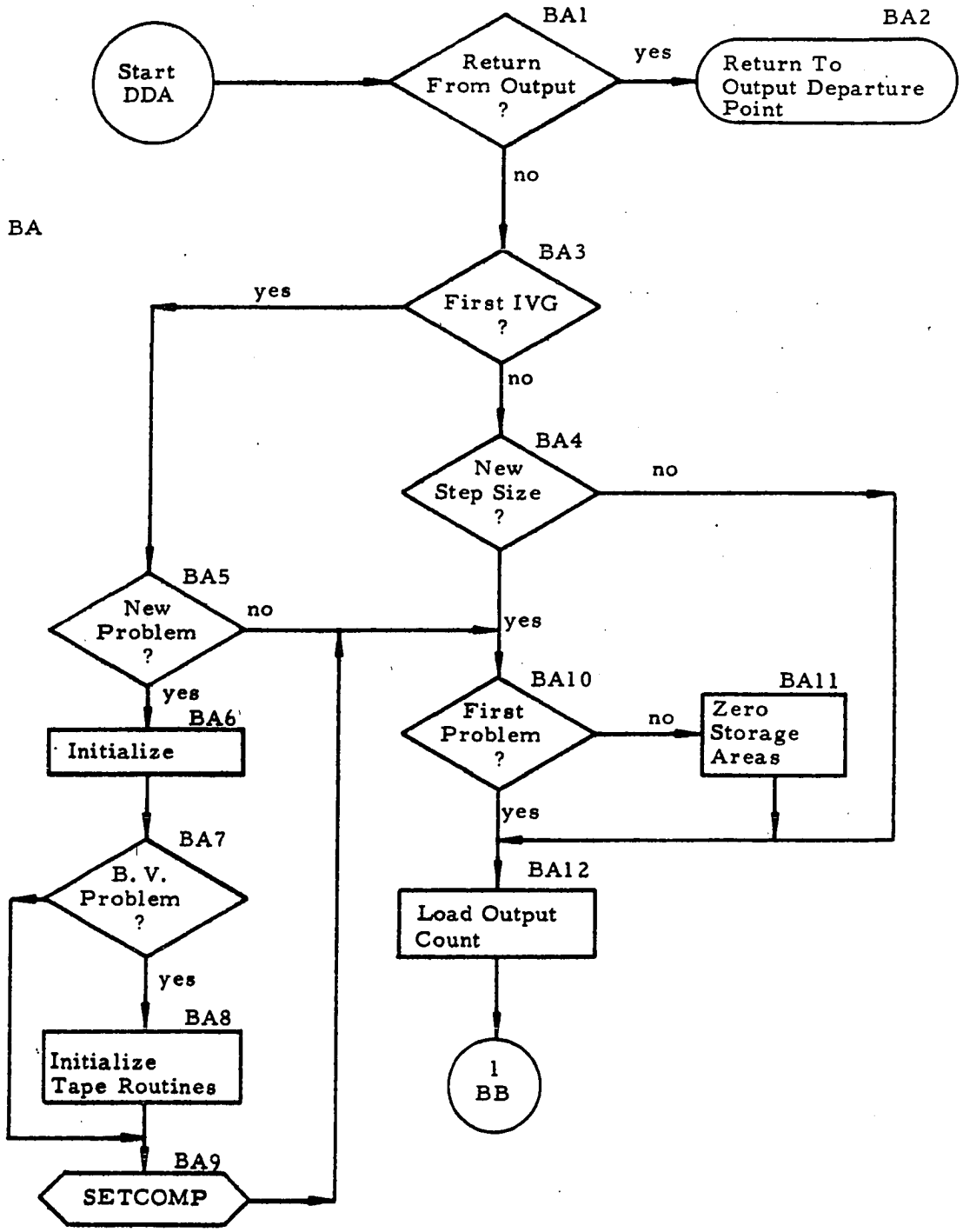


Figure 19. Flow chart for first portion of DDA subprogram

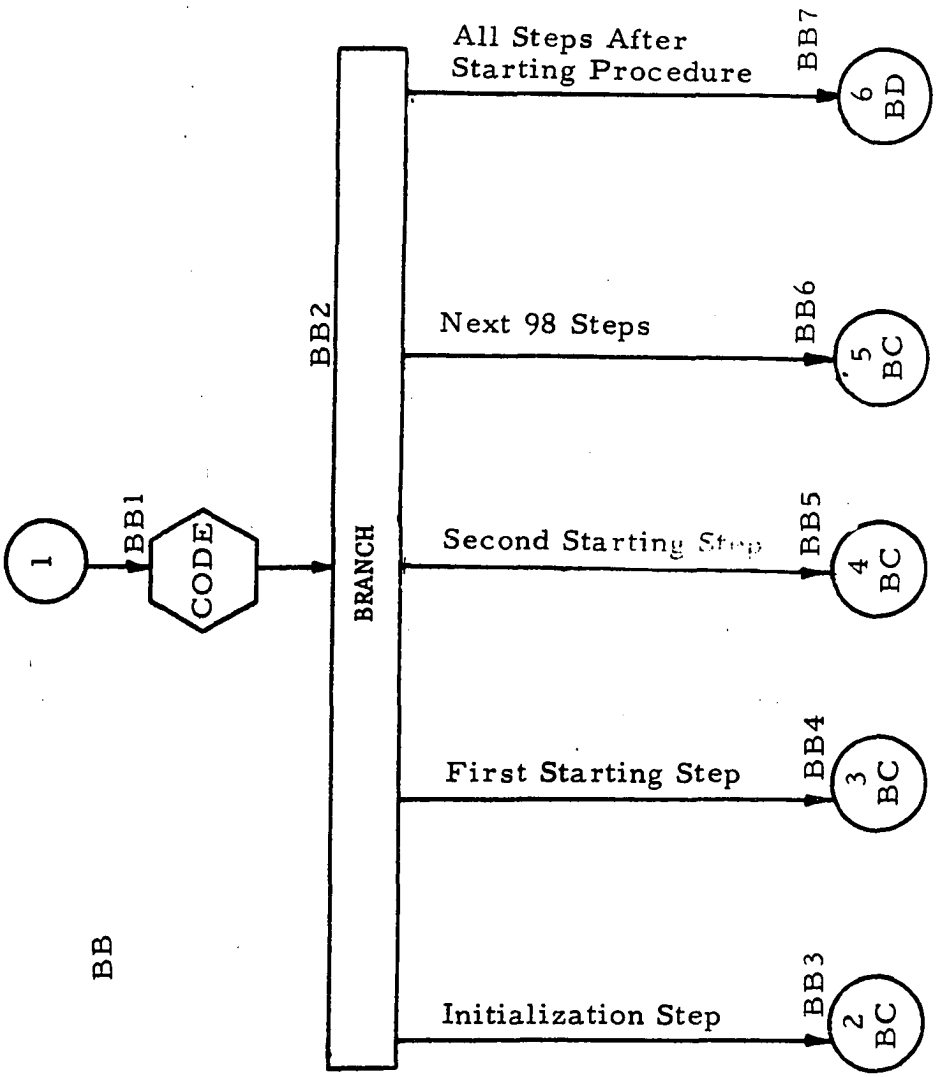


Figure 20. Flow chart for BRANCH portion of DDA subprogram

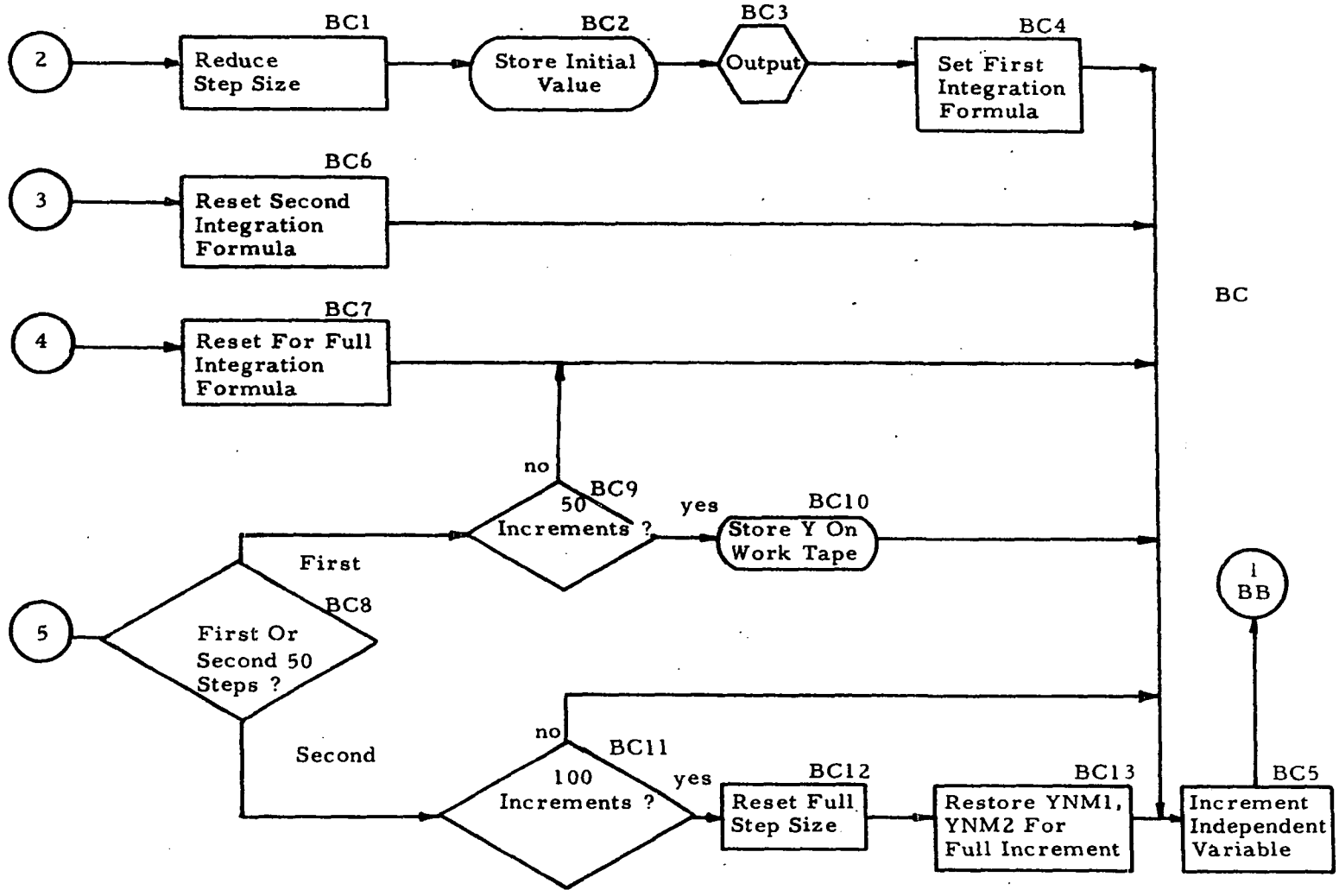


Figure 21. Flow chart for integration step control portion of DDA subprogram

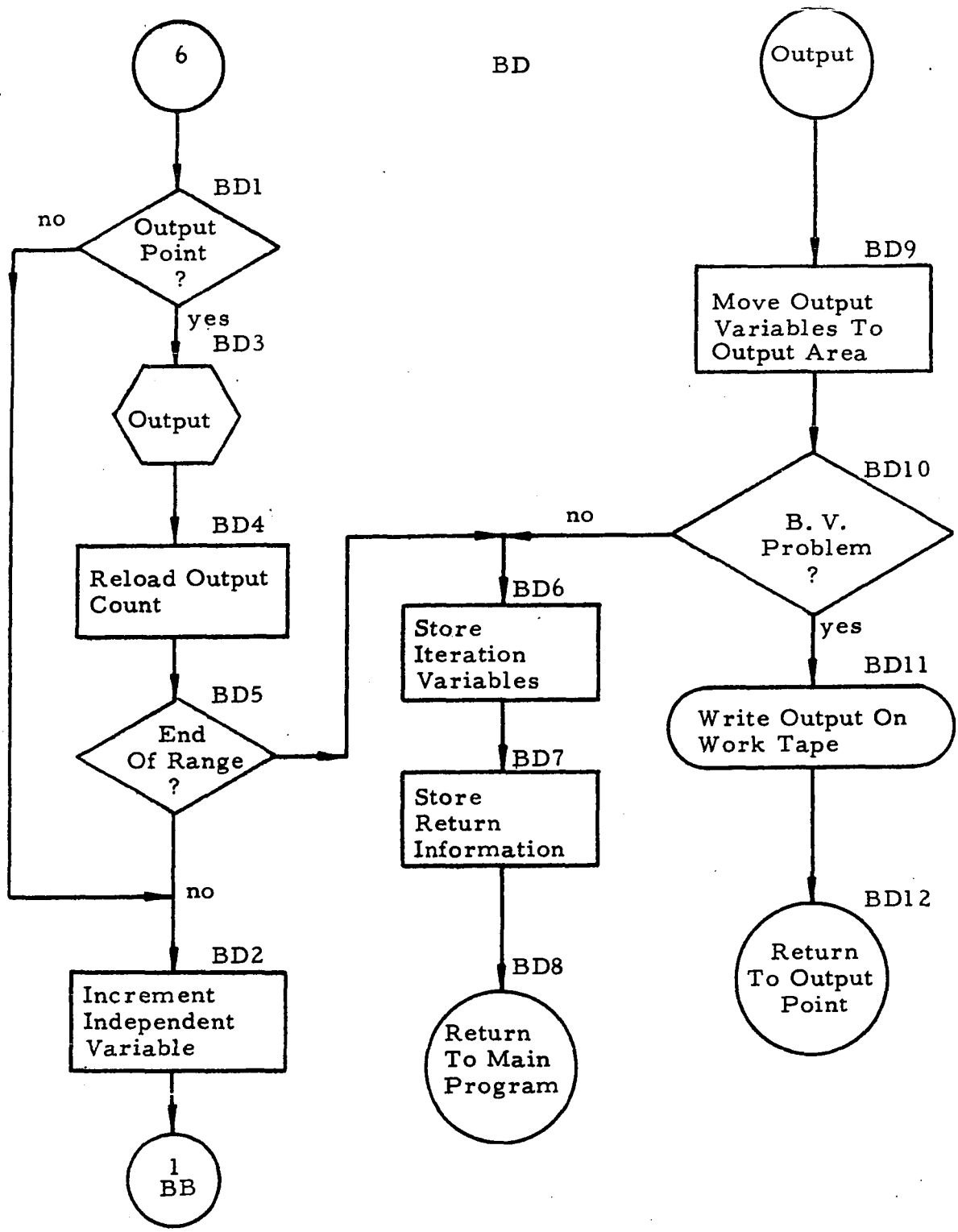


Figure 22. Flow chart for end portion of DDA subprogram

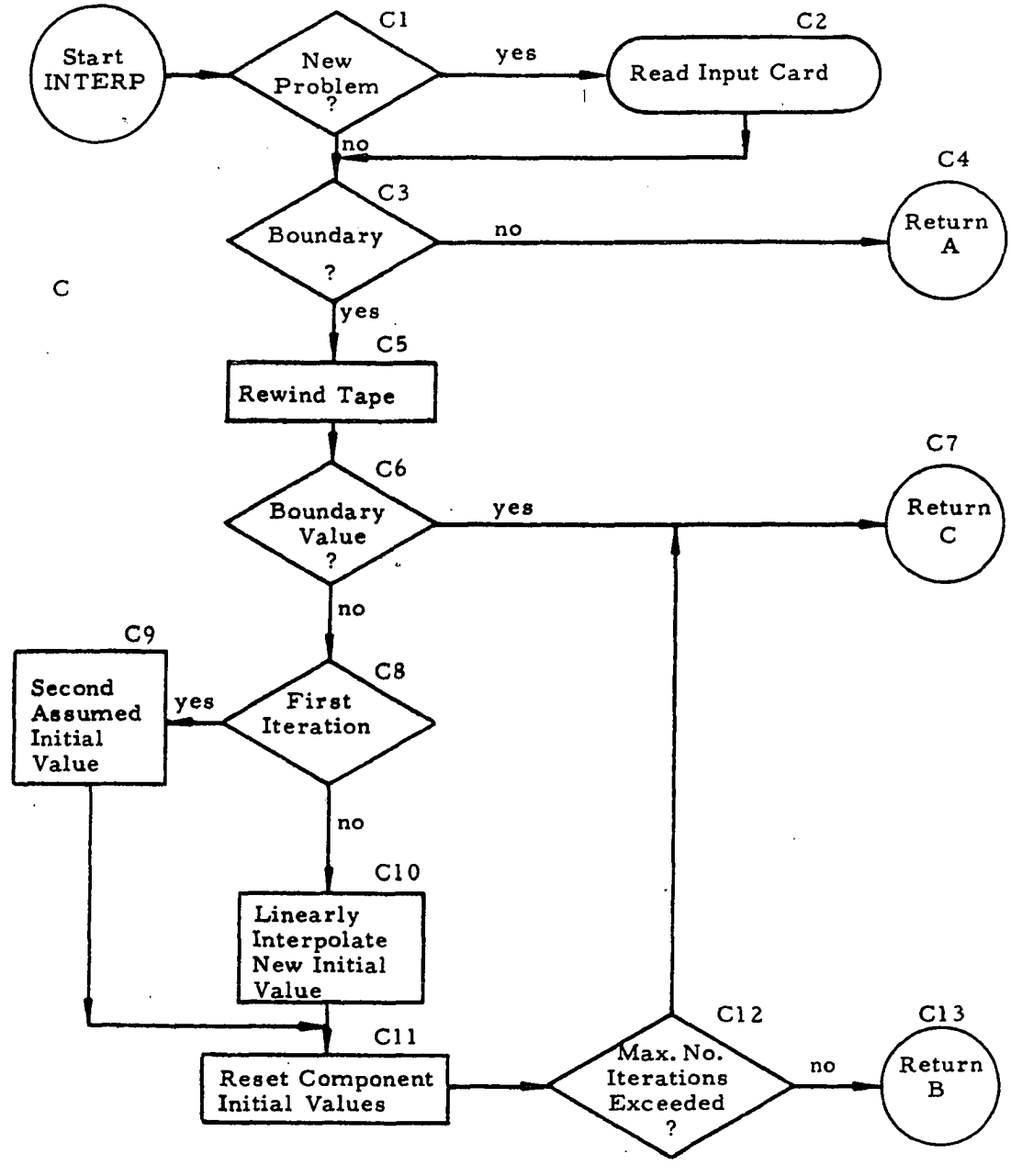


Figure 23. Flow chart for INTERP subprogram

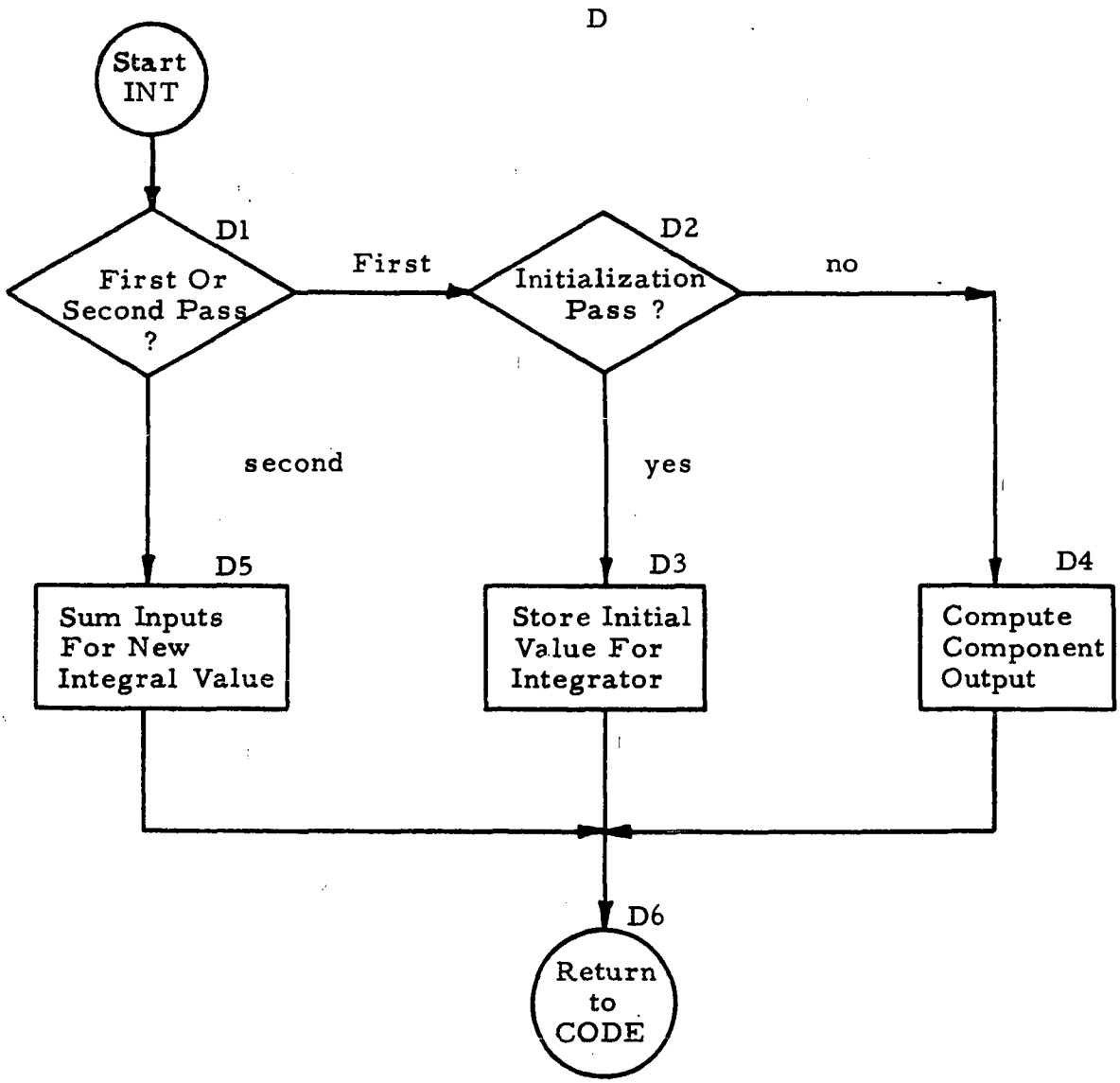


Figure 24. Flow chart for integrator subroutine

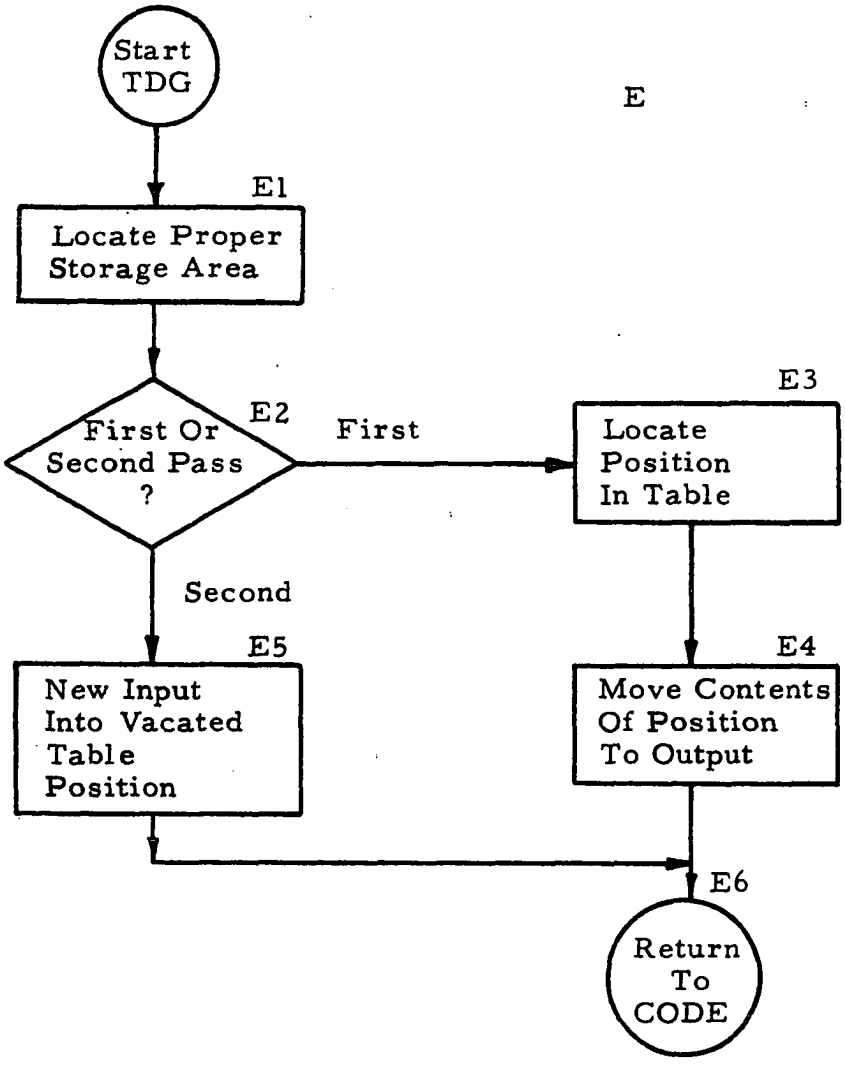


Figure 25. Flow chart for transport delay generator subroutine

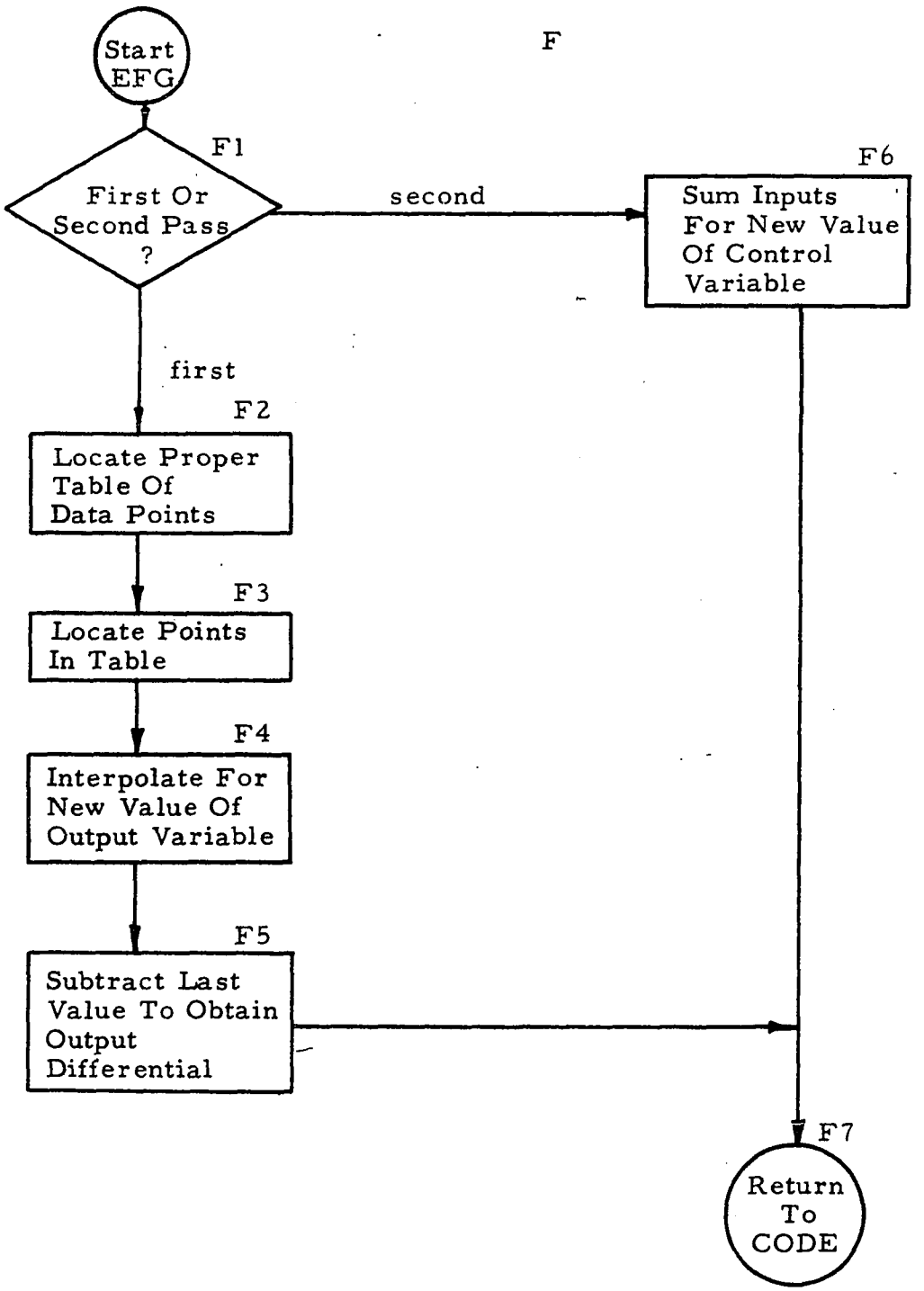


Figure 26. Flow chart for empirical function generator subroutine

COMMON storage area designated "CODE". Data for empirical function generators are read into proper storage area. All independent variable generator data are read in and stored.

Block A3: Locations of desired output variables, and output format are read in.

Block A4: The first independent variable generator is moved into the working area.

Block A5: Control is transferred to the DDA subprogram, which carries out the computations.

Block A6: Control may be returned to the main program for several reasons shown in subsequent blocks. In block A6 a check is made to see if an output point has been reached.

Block A7: Output variables are written out and control transferred back to the DDA subprogram.

Block A8: A check is made to see whether control was returned to the main program because of an error condition.

Block A9: If a boundary value problem is being solved, control is transferred to the INTERP subprogram; otherwise this block is skipped.

Block A10: A check is made to see if the integration limits are to be extended.

Block A11: A new independent variable generator is moved into the working area.

Block A12: If another iteration is required for a boundary

value problem, the problem is re-started with the first independent variable generator.

Block A13: A check is made to see if the boundary value problem work tape must be processed.

Block A14: During each iteration the output for a boundary value problem is stored on a work tape. If a new iteration is required, the tape will be re-wound by the INTERP subprogram and the incorrect data discarded. After the boundary condition has been matched or when an error condition occurs this work tape is processed by the main program to produce the desired output.

Block A15: If this is a multi-file run then the next problem is started; otherwise the computations are finished.

2. Flow chart BA, Figure 19; DDA subprogram

Block BA1: When control is transferred to this subprogram it must determine the current status of the problem solution. The first check is to determine whether this is merely a return after writing an output point.

Block BA2: After an output data point has been written, control is returned to the block which originally transferred control to the output routine. (BC3 or BD3)

Block BA3: A check is made to determine if this is the first independent variable generator for the problem.

Block BA4: If this is not the beginning of the problem, then a check is made to see if the integration increment has been changed.

Block BA5: If the first independent variable generator is being used, then a check is made for a new problem or a new iteration for a boundary value problem.

Block BA6: A new problem is initialized.

Block BA7-BA8: For a new boundary value problem, parts of the subprogram involving magnetic tape procedures must be initialized.

Block BA9: "SETCOMP" subroutine uses the information contained on the component input cards to compile the block of instructions comprising the subroutine "CODE" which establishes linkages with the computational subroutines and, in effect, establishes component interconnections.

Block BA10-BA11: For the first problem of a run the core storage areas will already be zeroed but for subsequent problems in a multifile run some areas must be reset to zero between problems.

Block BA12: Output points are determined by the step size and the number of steps in each output interval.

This information is obtained from the current independent variable generator and is reset before computations are started under control of subroutine "CODE" in block BB1.

3. Flow chart BB, Figure 20; DDA subprogram

Block BB1: "CODE" is a subroutine compiled by an initialization routine for each problem. It consists of a block of machine-language instructions and data representing each of the components. The first instruction loads an index register with the serial number of the component and the second instruction causes a branch to the appropriate computational subroutine. The index containing the component serial number is used by the subroutine in determining the locations within the working storage areas of the five working registers associated with the component. The remaining words in the coding block contain the information required by the subroutine to carry out the computations. The last instruction of "CODE" causes a return to "BRANCH" after all components have been updated.

Block BB2: "BRANCH" is a branch point which causes control to be transferred to the appropriate routine, depending upon the status of the computations.

Block BB3: The first pass through "CODE" initializes all of the components.

Block BB4: The first starting step in the integration procedure uses a reduced increment size and uses the first starting formula.

Block BB5: On the second step the second integration formula is used.

Block BB6: The first two integration steps are completed using the reduced increment size and the full integration formula.

Block BB7: Subsequent integration steps are made using the full increment size.

4. Flow chart BC, Figure 21; DDA subprogram

Block BC1: To compensate for the reduced accuracy of the starting integration formulas the increment size is reduced to 1/50 of the full step size specified by the independent variable generator.

Block BC2: After one hundred of the reduced steps have been taken, the full integration formula is used with the full step size. The initial values of the integrators must be stored on a work tape to be used later as the points YNM2 for the full size steps.

Block BC3: The initial values of the output variables are written out.

Block BC4: For the first reduced size integration step the first starting formula is used.

Block BC5: The independent variable is incremented before control is returned to "CODE" for the next step.

Block BC6: For the second starting step in the integration the second starting formula is used.

Block BC7: After the second starting step the full integration formula using the two past points is used with reduced size increments.

Block BC8: The program branches, depending upon whether the starting procedure is currently in the first or second fifty steps of the integration with the reduced increments.

Block BC9-BC10: After fifty of the reduced increments have been taken the integrated values held in the Y register of the integrators are stored on a work tape to be used later as the points YNM1 for each integrato. with the full step size for the integration increments.

Block BC11-BC12: After one hundred reduced increments have been completed, the increment size is reset to the full value specified by the independent variable generator.

Block BC13: The full integration formula requires two previous points, YNM1 and YNM2. These points were stored on a work tape for each integrator as the

initial values and as the integrated values of the output variables after fifty reduced increment steps. They are now read from the work tape into the YNM1 and YNM2 registers of the integrators to be used in the integration formulas for the full increment size.

5. Flow chart BD, Figure 22; DDA subprogram

Block BD1: After each integration step a check is made to determine whether an output point has been reached.

Block BD2: If an output point has not been reached, the independent variable generator is incremented and control is returned to "CODE" to continue the computations.

Block BD3: At an output point control is transferred to the output routine.

Block BD4: After output, control is returned to this block, which reloads the output departure count.

Block BD5: After each output a check is made to see if the upper limit of integration specified by the IVG (independent variable generator) has been reached.

Block BD6: Before control is transferred back to the Fortran program, the current and previous values of the control variable for a boundary value problem

are moved into a COMMON storage area for use by the INTERP subprogram.

Block BD7: The settings of electronic switches and the values of certain index words must be stored for use if control is transferred back to the DDA subprogram by the main Fortran program.

Block BD8: Control is transferred back to the Fortran program to process an output point, to check for a new IVG, to reset for a new iteration, or to initialize and read input for a new problem.

Block BD9: When an output point has been reached, this block moves the current values of the output variables to the output storage area.

Block BD10: If a boundary value problem is not being solved the output point will be processed by the Fortran program and control returned directly to the output departure point.

Block BD11: If a boundary value problem is being solved, the output point will be stored on a work tape for processing after the problem has been finished.

Block BD12: Priority tape operations are used for writing the work tape and computations are continued during this output. At the completion of the tape operation, it is checked for validity after which computations are resumed at the point where they were interrupted.

6. Flow chart C, Figure 23: INTERP subprogram

Block C1: A check is made for a new problem.

Block C2: For a new problem, the input card with the control parameters is read in.

Block C3: A check is made to see if the desired boundary has been reached.

Block C4: If the boundary has not yet been reached, control is returned to the main program for a new IVG.

Block C5: If the boundary has been reached the work tape is rewound.

Block C6: The current value of the control variable is compared with the desired value.

Block C7: If the problem is finished control is returned to the main program to process the output from the work tape.

Block C8: If another iteration is required then a check is made to see if the iteration just finished was the first iteration.

Block C9: For the second iteration a second assumed initial value of the control variable is used.

Block C10: For subsequent iterations the results of the past two iterations are used to linearly interpolate for a new initial value for the control variable.

Block C11: Initial values of the components are reset where necessary. The statements required to do this depend upon the problem. This makes it necessary to rewrite and recompile this subroutine for each boundary value problem (see Section V, Example problem 2).

Block C12: The solution may not converge to the desired boundary value. Therefore, if, after a specified number of iterations, the desired value has not been reached the problem is discontinued and the output from the last iteration is processed.

Block C13: If another iteration is required, control is transferred back to the Fortran program to restart the problem with the new initial conditions.

7. Flow chart D, Figure 24; Subroutine for integrators

Block D1: Two passes are made through each of the components in updating them. A check is made here for the first or second pass.

Block D2: For the first pass, a check is made for the initialization step.

Block D3: In the initialization step the initial value for each integrator, which was specified on its input card, is stored as the current value of the

integrated variable.

Block D4: During the first pass in updating an integrator the output of the integrator is computed from the current value of the integrated variable and the two preceeding values.

Block D5: On the second pass the new value of the integrated variable is obtained by summing the inputs for the component and adding this sum to the last integrated value.

Block D6: In updating the components control is switched back and forth between the individual instruction blocks in "CODE" and the computation subroutines. After an integrator has been updated it transfers control to the first instruction of the next component block in "CODE".

8. Flow chart E, Figure 25; Subroutine for transport delay generators

Block E1: Each transport delay generator uses a separate block of 100 storage words. The location of the proper storage area for this particular component is determined.

Block E2: A check is made for the first or second pass through the component.

Block E3-E4: During the first pass the location of the output value within the storage block is determined from the delay time specified for the component,

and this value is moved to the output register of this delay generator.

Block E5: On the second pass the input to the delay generator is stored in the location from which the output was moved during the first pass. This value will become the output after the number of time increments specified for the delay.

Block E6: Control is returned to the first instruction of the next component in "CODE".

9. Flow chart F, Figure 26; Subroutine for empirical function generators

Block F1: A check is made for the first or second pass through the component.

Block F2: Each EFG (empirical function generator) uses a specific block of 100 storage words. This routine locates the proper storage area.

Block F3: Either linear or parabolic interpolation may be used between the data points. This routine obtains the point preceeding the desired value and two points following it from the table.

Block F4: Linear or parabolic interpolation is used to determine the value of the output variable corresponding to the current value of the input variable.

Block F5: The previous value of the output variable is subtracted from the current value to obtain the output differential.

Block F6: On the second pass the inputs to the component are summed and added to the previous value of the input variable to obtain its new value.

Block F7: Control is returned to the next component instructions in "CODE".

V. EXAMPLE PROBLEMS

A. Purpose

Since the purpose of this project was to develop the programming system described in this dissertation as a general computational device rather than to apply it to a specific engineering problem, the detailed discussion of a specific large problem is not included in the dissertation. However, during the development of the system some relatively simple problems were used as test problems for checking the operation of the system whenever changes in, or additions to the system were made. Two such problems are presented here for the purpose of illustrating, in detail, the use of the system.

B. Example 1. Initial Value Problem

The two equations given on pages 20 and 25 represent mass and energy balances during the transient operation of a continuous stirred-tank chemical reactor in which a first order exothermic reaction is occurring, the heat of reaction being removed by cooling coils. The variables C and T represent concentration and temperature. The independent variable is time. The transient energy and mass balance equations for this problem have been derived by Amundson and Bilous (34) and the derivation will not be repeated here since the method of solution is of primary interest.

The system is shown in Figure 27. An initial value problem based on this system consists of determining the temperature and concentration as a function of time, from start-up to steady-state, when the initial temperature and concentration are known. The computer diagram for this problem is shown in Figures 8 and 9.

Figure 28 is the input listing for the problem. Only one independent variable generator is used specifying integration limits of 0 to 3000 seconds, and increment size of 10 seconds, and output every 300 seconds (30 increments). Output variables are concentration and temperature, which are the input variables for integrators number 17 and 12. Figure 10 is the input card for integrator 17. The output for the problem is given in Figure 29. Solution of the problem required about 30 seconds of computer time.

C. Example 2. Boundary Value Problem

The stirred tank reactor problem of Example 1 may be used to illustrate the use of the DIAN system for solving boundary value problems. In this case suppose that one wishes to determine the initial concentration which would produce a given steady-state concentration. Although the differential equations reduce to algebraic equations at steady-state, they cannot be solved explicitly for temperature and concentration since the equations are transcendental.

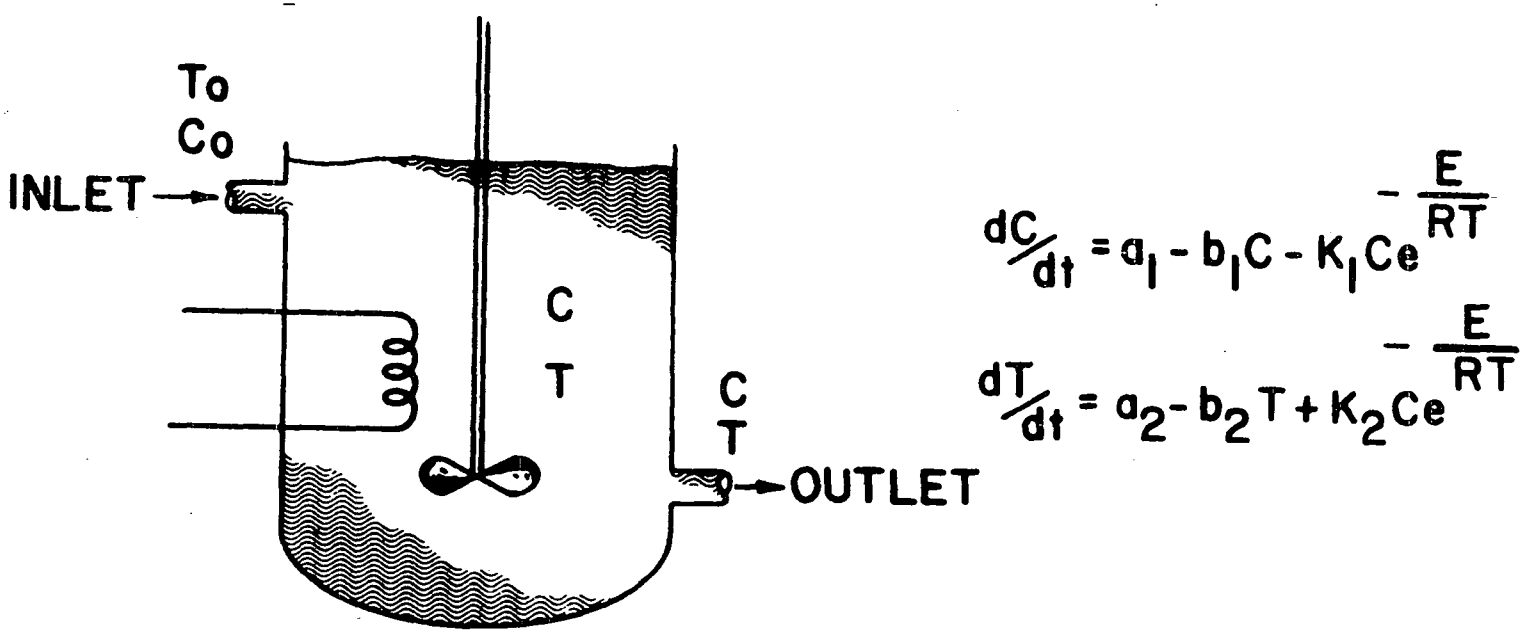
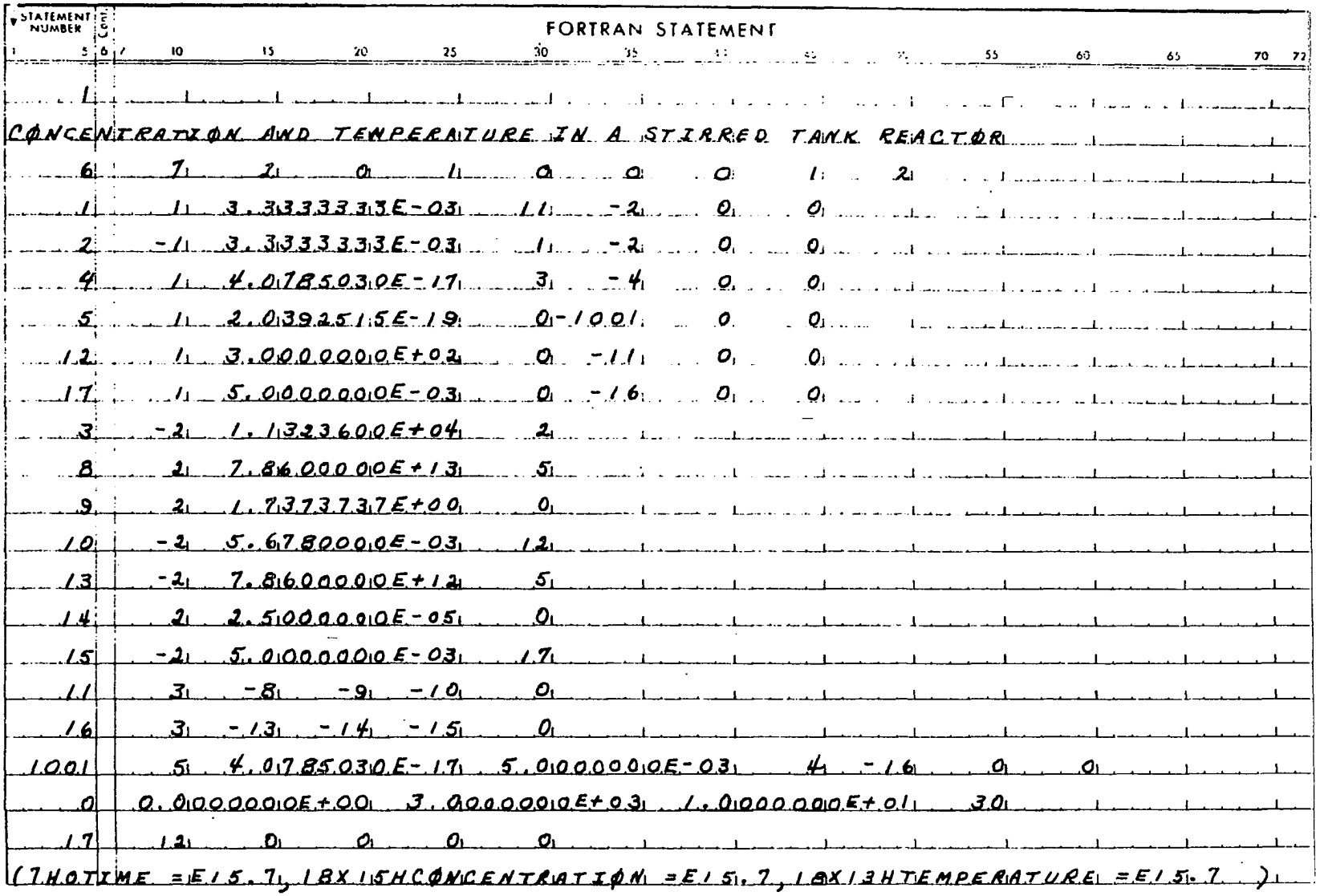


Figure 27. Transient operation of a stirred tank reactor



```
1
CONCENTRATION AND TEMPERATURE IN A STIRRED TANK REACTOR
6      7     2     0     1     0     0     0     1     2
1      1  3.333333E-03     1    -2     0     0
2     -1  3.333333E-03     1    -2     0     0
4      1  4.017850E-17     3    -4     0     0
5      1  2.039251E-19     0  -1001     0     0
12     1  3.000000E+02     0    -11     0     0
17     1  5.000000E-03     0    -16     0     0
3     -2  1.132360E+04     2
8      2  7.860000E+13     5
9      2  1.737373E+00     0
10    -2  5.678000E-03     12
13    -2  7.860000E+12     5
14     2  2.500000E-05     0
15    -2  5.000000E-03     17
11     3    -8    -9   -10     0
16     3   -13   -14   -15     0
1001   5  4.017850E-17  5.000000E-03     4    -16     0     0
0      0.000000E+00  3.000000E+03  1.000000E+01     30
17     12    0     0     0     0
(7HOTIME =E15.7,18X15HCONCENTRATION =E15.7,18X13HTEMPERATURE =E15.7)
```

Figure 28. Input listing for example problem 1

CONCENTRATION AND TEMPERATURE IN A STIRRED TANK REACTOR

TIME = 0.0000000E 00	CONCENTRATION = 0.5000000E-02	TEMPERATURE = 0.3000000E 03
TIME = 0.3000000E 03	CONCENTRATION = 0.4635067E-02	TEMPERATURE = 0.3048973E 03
TIME = 0.6000000E 03	CONCENTRATION = 0.4473963E-02	TEMPERATURE = 0.3057898E 03
TIME = 0.9000000E 03	CONCENTRATION = 0.4425432E-02	TEMPERATURE = 0.3059524E 03
TIME = 0.1200000E 04	CONCENTRATION = 0.4413106E-02	TEMPERATURE = 0.3059820E 03
TIME = 0.1500000E 04	CONCENTRATION = 0.4410234E-02	TEMPERATURE = 0.3059874E 03
TIME = 0.1800000E 04	CONCENTRATION = 0.4409597E-02	TEMPERATURE = 0.3059884E 03
TIME = 0.2100000E 04	CONCENTRATION = 0.4409461E-02	TEMPERATURE = 0.3059885E 03
TIME = 0.2400000E 04	CONCENTRATION = 0.4409433E-02	TEMPERATURE = 0.3059886E 03
TIME = 0.2700000E 04	CONCENTRATION = 0.4409427E-02	TEMPERATURE = 0.3059886E 03
TIME = 0.3000000E 04	CONCENTRATION = 0.4409426E-02	TEMPERATURE = 0.3059886E 03

Figure 29. Output for example problem 1

The problem can be solved using the DIAN system by solving the differential equations as a boundary value problem. Figure 30 is the listing for the subprogram INTERP which is used in solving this problem. The input listing for the problem is shown in Figure 31. The input deck is very similar to that used for the initial value problem of Example 1 except that several independent variable generators are used since it is necessary to check periodically to see whether steady-state has been reached. This is done by the INTERP subprogram, which is entered only at the end of the integration range of an independent variable generator. Also, one additional input card is required by the subprogram.

For a boundary value problem the control variable is specified by listing it as the first output variable. In this case, concentration is specified as the control variable by listing integrator 17 as the location of the first output variable.

In this example, a steady-state concentration of 5.0×10^{-3} is desired. To start the problem two initial values are assumed for the concentration. One of these (5.0×10^{-3}) is used on the input cards as the initial value for integrator number 17, as the constant for constant multiplier number 15, and as one of the initial values for function multiplier number 1001. It is also used indirectly for the constant in constant multiplier number 14 since the constant a_1 is a function of the initial concentration.

```
      SUBROUTINE INTERP
C
      DIMENSION IVG(5), CODE(7500), YOUT(48), OUT(48),FMT(14),
     1 TITLE(14), N(150), D(1500), B(7005), MAP(7500)
      COMMON CODE, IVG, NCOMP, NOUT, NEFG, NTDG, NABS, NMUL, NREL,
     1 NSUM, NCON, NINT, YOUT, OUT, XNM1, X, M7, M6, I2, I1, D, B,
     2 TITLE, M5, N, FMT
      EQUIVALENCE (CODE, MAP)
    1 FORMAT (4E15.7, I5)
      IF(I1 - LASTI1) 22, 4, 2
    2 M9 = 1
      M4 = 0
      READ INPUT TAPE 5, 1,  XI2, XFIN, DX, EPS, MAX
      LASTI1 = I1
    4 DELTA = ABSF(X-XNM1)
    5 IF(DELTA - EPS) 8, 8, 6
    6 M6 = 1
      RETURN
    8 REWIND 3
      D = ABSF(X - XFIN)
      IF(D - DX) 20, 20, 10
   10 GO TO (14, 12), M9
   12 XI2 = XI1 + ((XI1 - XI2) * (XFIN - XF1))/(XF1 - X)
   14 M9 = 2
      XF1 = X
C     THE FOLLOWING STATEMENTS DEPEND ON THE PROBLEM
      XI1 = CODE(38)
      CODE(38) = XI2
      CODE(69) = XI2
      CODE(86) = XI2
      CODE(65) = XI2/200.0
C
      M4 = M4 + 1
      IF(MAX - M4) 20, 20, 18
   18 M6 = 2
      RETURN
   20 M6 = 3
      RETURN
   22 STOP
      END
```

Figure 30. Listing for INTERP subprogram

```
BOUNDARY VALUE VERSION OF THE STIRRED TANK REACTOR PROBLEM
 6   7   2   0   1   0   0   0   8   2
 1   1  3.333333333E-03  1.1  -2   0   0
 2  -1  3.333333333E-03  1   -2   0   0
 4   1  4.017850310E-17  3   -4   0   0
 5   1  2.039251515E-19  0  -100.1  0   0
 12  1  3.000000010E+02  0   -1.1  0   0
 17  1  5.000000010E-03  0   -16   0   0
 3  -2  1.132360010E+04  2
 8   2  7.816000000E+13  5
 9   2  1.737373737E+00  0
 10 -2  5.678000010E-03  12
 13 -2  7.816000000E+12  5
 14  2  2.510000010E-05  0
 15 -2  5.000000010E-03  17
 11  3  -8   -9   -10  0
 16  3  -13  -14  -15  0
1001 5  4.017850310E-17  5.010000010E-03  4  -16  0   0
 0   0.000000000E+00  5.000000010E+02  5.010000000E+00  50
 0   5.010000010E+02  1.010000000E+03  1.010000010E+01  25
 0   1.000000010E+03  1.510000010E+03  1.010000000E+01  25
 0   1.510000000E+03  2.000000010E+03  1.010000010E+01  25
 0   2.000000010E+03  2.500000010E+03  1.000000010E+01  25
 0   2.510000000E+03  3.000000010E+03  1.000000010E+01  25
 0   3.000000010E+03  3.500000010E+03  2.000000010E+01  20
 0   3.500000010E+03  4.000000010E+03  2.000000010E+01  20
 17  12   0   0   0   0
(THOTIME =E15.7,18X,16HCONCENTRATION =E15.7,18X,13HTEMPERATURE =E15.7 )
 5.500000010E-03  5.010000010E-03  1.000000010E-08  1.000000010E-10  10
```

Figure 31. Input listing for example problem 2

The second assumed initial concentration (5.5×10^{-3}) is specified on the last input card, along with the desired steady-state concentration (5.0×10^{-3}), plus limits for use in determining when steady-state has been reached and when the boundary condition has been met. For measuring the approach to steady-state the current value of the control variable and its value at the end of the preceeding integration step are automatically supplied to the INTERP subprogram by the DDA subprogram. The input data card for this problem also contains the number 10 which specifies that if the boundary condition has not been met after 10 iterations, the problem should be terminated and the results of the last iteration printed out.

After two steady-state solutions have been computed using the two assumed initial concentrations, a linear interpolation procedure is used to compute a new trial initial concentration.

Before starting each new iteration the initial values of the components mentioned above (integrator 17, constant multiplier 15, function multiplier 1001, constant multiplier 14) must be reset with the new initial values. The numbers which must be changed before starting the next iteration are located somewhere in the storage block called CODE. To determine the location of an initial value for a given component the following formulas may be used.

(1) Initial Values of Integrators:

$I = (7 \times NPI) + 3$

where: NPI = number of integrators preceeding the desired integrator in the input deck

I = the location within CODE, i.e., CODE (I) is the symbolic storage location of the desired initial value.

(2) Constants for Constant Multipliers:

$I = (7 \times NINT) + (4 \times NPC) + 3$

where: NINT = total number of integrators in the input deck

NPC = number of constant multipliers preceeding this component in the input deck

(3) Initial Values for Relay Control Variables:

$I = (7 \times NINT) + (4 \times NCON) + (6 \times NSUM) + (7 \times NPR) + 3$

where: NCON = total number of constant multipliers in the input deck

NSUM = total number of summers in the input deck

NPR = number of relays preceeding this component in the input deck

(4) Initial Values for Function Multipliers:

$$I = (7 \times NINT) + (4 \times NCOM) + (6 \times NSUM) + (7 \times NREL) + (8 \times NPR) + K$$

where: NREL = total number of relays in the input deck

NPM = number of function multipliers preceeding this component in the input deck

K = 3 for U_o

K = 4 for V_o

(5) Initial Values for Empirical Function Generators:

$$I = (7 \times NINT) + (4 \times NCON) + (6 \times NSUM) + (7 \times NREL) + (8 \times NMUL) + (3 \times NABS) + (4 \times NTDG) + (6 \times NPE)$$

where: NMUL = total number of function multipliers in the input deck

NABS = total number of absolute value generators in the input deck

NTDG = total number of transport delay generators in the input deck

NPE = number of empirical function generators preceeding this component in the input deck

This example boundary value problem represents one form in which a boundary condition might occur; that is, as the

steady-state value of some variable. Another way in which a boundary condition occurs is as a specified value of a control variable at a given value of the independent variable (e.g. temperature at the wall of a pipe). The Fortran program for the INTERP subprogram given in Figure 30 may be modified easily for such a problem. The input variable EPS would be replaced by TFIN (the final value of the independent variable) and statements 4 and 5 would be replaced by:

```
4   Y = IVG(3)
5   IF (TFIN - Y) 8,8,6
```

The subprogram INTERP as listed in Figure 30 is for use with the 20K system. To use it with the 10K system the DIMENSION statement must be changed to correspond to the DIMENSION statement of the main program of the 10K system.

The output for Example Problem 2 is given in Figure 32. Solution of this problem required about 90 seconds of computer time.

BOUNDARY VALUE VERSION OF THE STIRRED TANK REACTOR PROBLEM

TIME = 0.0000000E 00	CONCENTRATION = 0.1528728E-01	TEMPERATURE = 0.3000000E 03
TIME = 0.2500000E 03	CONCENTRATION = 0.5247677E-02	TEMPERATURE = 0.3045361E 03
TIME = 0.5000000E 03	CONCENTRATION = 0.5017152E-02	TEMPERATURE = 0.3056331E 03
TIME = 0.7500000E 03	CONCENTRATION = 0.5003718E-02	TEMPERATURE = 0.3058985E 03
TIME = 0.1000000E 04	CONCENTRATION = 0.5000897E-02	TEMPERATURE = 0.3059628E 03
TIME = 0.1250000E 04	CONCENTRATION = 0.5000215E-02	TEMPERATURE = 0.3059784E 03
TIME = 0.1500000E 04	CONCENTRATION = 0.5000050E-02	TEMPERATURE = 0.3059822E 03
TIME = 0.1750000E 04	CONCENTRATION = 0.5000010E-02	TEMPERATURE = 0.3059831E 03
TIME = 0.2000000E 04	CONCENTRATION = 0.5000000E-02	TEMPERATURE = 0.3059833E 03
TIME = 0.2250000E 04	CONCENTRATION = 0.4999998E-02	TEMPERATURE = 0.3059834E 03
TIME = 0.2500000E 04	CONCENTRATION = 0.4999997E-02	TEMPERATURE = 0.3059834E 03

Figure 32. Output for example problem 2

VI. CONCLUSIONS AND SUMMARY

In its present form DIAN is a versatile programming system which can serve many useful purposes. It combines the advantages of a digital computer with the ease of programming an analog computer. Less programming time is required than for a conventional digital computer program and a problem may be altered easily without additional programming or compilation time.

Since DIAN operates in basic machine language and does not require recompilation for each problem, it is more efficient than either an interpretive or compiler method of simulating analog or DDA components. While the program may not be as efficient as a well written conventional digital computer program in solving a particular problem, the total computer time will probably be reduced due to the elimination of the compilation step. Since it is compatible with Fortran, DIAN may be used as a part of a larger Fortran program. Expanding the system requires only that the Fortran portion be recompiled. The computation subroutine need never be recompiled unless extensive changes are made such as the addition of other types of components. One relatively simple expansion of the system can be made in this way by adding provisions for a library subroutine controlling the I.B.M. 1403 to obtain graphical output.

The system may be used to simulate an analog computer or DDA or, it may be considered as a separate technique of problem solution with its own unique advantages. A problem may be started on an analog computer and, as the work progresses and refinements are added to the mathematical model, the program may become too large for the available analog computer facilities. If a digital computer is available, this system may be used without the necessity of completely rewriting the problem in the form of a conventional digital computer program. Alternatively, an initial program may be simplified so that it can be set up on an available analog computer for initial study. After this, the simplifications may be removed and the complete problem solved on the digital computer without extensive reprogramming. Such an approach permits rapid variation of parameters on the simplified model using the analog computer to define the range of interest for each variable.

The advantages of this system over conventional analog computers are:

(1) Use of floating point arithmetic eliminates the problem of scaling.

(2) The capacity of the system is much larger than that of most analog computers.

(3) More general types of boundary value problems may be solved than may be handled on an analog computer without memory.

(4) Any degree of accuracy may be achieved, with a corresponding increase in computing time, by reduction of the integration increment size.

(5) Time lag due to material transport delay may be represented much more accurately than on an analog computer.

(6) Output may be obtained in either graphical or tabular form.

(7) Many non-linear functions may be generated accurately. This results from the use of a more general type of integrator than the analog computer integrator.

Several other systems, cited in section II, have been developed to simulate the operation of analog computers or digital differential analyzers. DIAN has several advantages over all of these systems.

(1) Use of a digital program in basic machine language eliminates interpretation time and the compilation step, and thereby, reduces computing time.

(2) DIAN may be used for solving more general types of boundary value problems.

(3) No special programming language is required since input is in the form of data cards of standard format.

(4) Any number of problems may be solved sequentially in a multi-file run.

(5) It is possible to expand or modify the system by additional Fortran programming or to incorporate it into a larger Fortran program.

(6) Non-linear functions may be generated by the use of the basic components. This is an advantage which conventional DDAs have over analog computers resulting from the use of a more general type of integrator. Of the other simulation systems, only DIDAS (19) possesses this feature since it also uses the DDA integrator.

Other important features of the system are:

(1) Integration increment size and output frequency may be changed at any predetermined time during the problem solution.

(2) Output format may be varied since the format is supplied as input information with each problem.

It was originally intended to design a system for simulating a DDA on the I.B.M. 7074 for use as a supplement to a small analog computer for solving problems which were too large for the analog computer. However, the system, in its present form, possesses sufficient advantages over analog computers to justify its use even for small problems. Also, it is applicable to types of problems which cannot be solved on ordinary analog computers.

VII. LITERATURE CITED

1. Bush, V. and A. H. Caldwell. A new type of differential analyzer. Franklin Institute Journal 240: 255-326. 1945.

2. Jackson, A. S. Analog computation. New York, McGraw-Hill Book Co. 1960.

3. Sprague, R. E. Fundamental concepts of the DDA method of computation. Mathematical Tables and Other Aids to Computation 6: 41-49. 1952.

4. Computing machines. Mechanical Engineering 73: 325-327. 1951.

5. Forbes, G. F. Digital differential analyzers. 4th ed. 13745 Eldridge, Sylmar, Calif., Author. 1957.

6. Mitchell, J. M. and S. Rukman. The TRICE ---- A high speed incremental computer. Institute of Radio Engineers National Convention Record 6, Part 4: 206-215. 1958.

7. Beck, R. M. Digital differential analyzer. Instruments and Automation 31: 1836-1837. 1958.

8. Bradley, R. E. and J. F. Genna. Design of a one-megacycle iteration rate DDA. Spring Joint Computer Conference Proceedings 21: 353-364. 1962.

9. King, E. M., Jr. and R. Gelman. Experience with hybrid computation. Fall Joint Computer Conference Proceedings 22: 36-43. 1962.

10. McLeod, J. Ten years of computer simulation. Institute of Radio Engineers Transactions on Electronic Computers EC11: 2-6. 1962.

11. Skramstad, H. K. Combined analog-digital techniques in simulation. In Alt, F. L. and M. Rubinoff, eds. Advances in computers. Vol. 3. pp. 275-298. New York, Academic Press, Inc. 1962.

12. Wilson, A. N. Use of a combined analog-digital system for re-entry vehicle flight simulation. Eastern Joint Computer Conference Proceedings 20: 105-113. 1961.

13. Skramstad, H. K. A combined analog-digital differential analyzer. Eastern Joint Computer Conference Proceedings 16: 94-100. 1959.

14. Schmid, H. Combined analog-digital computing elements. Western Joint Computer Conference Proceedings 19: 299-312. 1961.

15. Selfridge, R. G. Coding a general-purpose digital computer to operate as a differential analyzer. Western Joint Computer Conference Proceedings 7: 82-84. 1955.

16. Lesh, F. H. Methods of simulating a differential analyzer on a digital computer. Association for Computing Machinery Journal 5: 281-288. 1958.

17. Lesh, F. H. and F. G. Curl. DEPI: an interpretive digital-computer routine simulating differential-analyzer operations. California Institute of Technology, Jet Propulsion Laboratory, Memorandum No. 20-141. 1957.

18. Curl, F. G. A comparison of computers. Communications and Electronics 80: 542-547. 1961.

19. Slayton, G. R. DIDAS, a digital differential analyzer simulator. Unpublished microfilmed paper. International Business Machines Corporation, Program Information Department, SHARE Library, Program No. 319. 1957.

20. Mendelson, M. J. The decimal digital differential analyzer. Aeronautical Engineering Review 13, No. 2: 42-54. 1954.

21. Hildebrand, J. B. Introduction to numerical analysis. New York, McGraw-Hill Book Co. 1956.

22. Milne, W. E. Numerical solution of differential equations. New York, John Wiley and Sons. 1953.

23. Hurley, J. R. and J. J. Skiles. DYSAC: a digitally simulated analog computer. Spring Joint Computer Conference Proceedings 23: 69-82. 1963.

24. Gaskill, R. A., J. W. Harris, and A. L. McKnight. DAS-A digital analog simulator. Spring Joint Computer Conference Proceedings 23: 83-90. 1963.

25. Stein, M. L., J. Rose, and D. B. Parker. A compiler with an analog-oriented input language. Western Joint Computer Conference Proceedings 15: 92-102. 1959.

26. Stein, M. L. Automatic digital programming of analog computers. Institute of Electrical and Electronics Engineers Transactions on Electronic Computers EC12: 100-111. 1963.

27. Stein, M. L. Changing from analog to digital programming by digital techniques. Association for Computing Machinery Journal 7: 10-13. 1960.

28. Green, C., H. D'Hoop, and A. Debroux. APACHE: a breakthrough in analog computing. Institute of Radio Engineers Transactions on Electronic Computers EC11: 699-706. 1962.

29. Palevsky, M. and J. V. Howell. Digital differential equation solver. Instruments and Control Systems 36: 118-121. 1963.

30. Silber, W. B. Function generation with a DDA. Instruments and Control Systems 33: 1895-1899. 1960.

31. Wierwille, W. W. A new method for obtaining continuous delays on the analog computer. Institute of Electrical and Electronics Engineers Transactions on Automatic Control AC8: 73-74. 1963.

32. Lapidus, L. Digital computation for chemical engineers. New York, McGraw-Hill Book Co. 1962.

33. Kunz, K. S. Numerical analysis. New York, McGraw-Hill Book Co. 1962.

34. Amundson, N. R. and O. Bilous. American Institute of Chemical Engineers Journal 1: 513-517. 1955.

VIII. ACKNOWLEDGMENT

The author wishes to express his appreciation to Dr. L. E. Burkhart for suggesting this project and for his advice and assistance, and to Dianne Farris for her help in preparing the manuscript and for her patience and encouragement during the entire period of graduate study.

IX. APPENDIX

A. Listings for 10K System

```
C    DIAN      FORTRAN MAIN PROGRAM - 10K SYSTEM
C
      DIMENSION  IVG(5), CODE(3000), YOUT(48), OUT(48),FMT(14),
     1 TITLE(14), N(150), D(1000), B(2505), MAP(3000)
      COMMON  CODE, IVG, NCOMP, NOUT, NEFG, NTDG, NABS, NMUL, NREL,
     1 NSUM, NCON, NINT, YOUT, OUT, XNM1, X, M7, M6, I2, I1, D, B,
     2 TITLE, M5, N, FMT
      EQUIVALENCE (CODE, MAP)
    1 FORMAT (2I5, E15.7, 4I5)
    3 FORMAT (2I5, E15.7, I5)
    5 FORMAT (6I5)
    7 FORMAT (2I5, 3E15.7, 2I5)
    9 FORMAT (I5, 3E15.7, I5)
   11 FORMAT (I5)
   13 FORMAT (14A5)
   15 FORMAT (2I5, E15.7, I5, A5, I5)
   17 FORMAT (3I5)
   19 FORMAT (4I5)
   21 FORMAT (10I5)
   23 FORMAT (1H1, 25X14A5)
   25 FORMAT (4E15.7)
   27 FORMAT (2I5, 2E15.7, 4I5)
      READ INPUT TAPE 5,11, NPROB
      DO541I=1, NPROB
      REWIND 2
      REWIND 3
      READ INPUT TAPE 5,13, (TITLE(L1), L1=1, 14)
      WRITE OUTPUT TAPE 10, 23, (TITLE(L1), L1=1,14)
      READ INPUT TAPE 5, 21, NINT, NCON, NSUM, NREL, NMUL, NABS, NTDG,
     1NEFG, NIVG, NOUT
      NCOMP =NINT + NCON + NSUM + NREL + NMUL + NABS + NTDG + NEFG
      K0 = NINT * 7
      K1 = K0 + 1
      K2 = K0 + (NCON * 4)
      K3 = K2 + 1
      K4 = K2 + (NSUM * 6)
      K5 = K4 + 1
      K6 = K4 + (NREL * 7)
      K7 = NOUT + 1
      K8 = 5 * NIVG
      K9 = K6 +1
      K10 = K6 + (NMUL * 8)
      K11 = K10 + 1
      K12 = K10 + (NABS * 3)
      K13 = K12 + 1
      K14 = K12 + (NTDG * 4)
      K15 = K14 + 1
```

Figure 33. Listing for Fortran main program

```
      K16 = K14 + 6
      K19 = (100 * NTDG) - 99
      IF(NINT) 56, 4, 2
    2 READ INPUT TAPE 5, 1, (CODE(K), K=1, K0)
    4 IF(NCON) 56, 8, 6
    6 READ INPUT TAPE 5, 3, (CODE(K), K=K1, K2)
    8 IF(NSUM) 56, 12, 10
   10 READ INPUT TAPE 5, 5, (CODE(K), K=K3, K4)
   12 IF(NREL) 56, 16, 14
   14 READ INPUT TAPE 5, 7, (CODE(K), K=K5, K6)
   16 IF(NMUL) 56, 20, 18
   18 READ INPUT TAPE 5, 27, (CODE(K), K = K9, K10)
   20 IF(NABS) 56, 24, 22
   22 READ INPUT TAPE 5, 17, (CODE(K), K = K11, K12)
   24 IF(NTDG) 56, 28, 26
   26 READ INPUT TAPE 5, 19, (CODE(K), K = K13, K14)
   28 IF(NEFG) 56, 34, 30
   30 DO32I4=1, NEFG
      READ INPUT TAPE 5, 15, (CODE(K), K=K15, K16)
      K19 = K19 + 100
      K20 = K19 - 1 + (2 * MAP(K16 - 2))
      READ INPUT TAPE 5, 25, (D(K), K=K19, K20)
      K15 = K15 + 6
   32 K16 = K16 + 6
   34 READ INPUT TAPE 5, 9, (N(K), K=1, K8)
      READ INPUT TAPE 5, 5, (YOUT(J1), J1 = 1, NOUT)
      READ INPUT TAPE 5, 13, (FMT(L3),L3=1, 14)
   36 DO46I2=1, NIVG
      DO38I3=1, 5
      L2 = (5 * I2) - 5 + I3
   38 IVG(I3) = N(L2)
   40 CALL DDA
      GO TO (42, 44, 48, 50, 46), M5
   42 WRITE OUTPUT TAPE 10, FMT, (OUT(L4), L4 = 1, K7)
      GO TO 40
   44 CALL INTERP
      GO TO (46, 36, 50), M6
   46 CONTINUE
   48 GO TO 54
   50 REWIND 3
      DO52I4=1, M7
      READ TAPE 3, (OUT(L4),L4=1, K7)
      WRITE OUTPUT TAPE 10, FMT, (OUT(L4),L4=1, K7)
   52 CONTINUE
   54 CONTINUE
   56 STOP
      END
```

Figure 33. (Continued)

```
AA00 EXECUTE   CNTRL7      DDA SUBPROGRAM FOR 10K SYSTEM
AA02           DC
AA04                @DDA        @
AA06                +0+0+325+0+9100000001
AA08 ORIGIN    CNTRL325,COUNTA
AA10           XZA   98,20
AA12           RS    98,IWSTORAGE
AA14           RG    98,SUBSTORE
AA16           ZA1   156
AA18           ZST1  TEMP
AA20           ZA1   TEMP+1
AA22           ZST1  102
AA24           ZA1   TEMP+2
AA26           ZST1  101
AA28           BES   12,0+X27
AA30           CD    I2(9),1
AA32           BE    DIAN
AA34           B     EXTENDY
AA36 OUTPUT    ZA1   Y
AA38           ZST1  OUT
AA40           XL    60,NOUT
AA42           XA    60,1
AA44           XLIN  61,YOUT-1+X60
AA46           ZA1   Y+X61
AA48           ZST1  OUT+X60
AA50           BIX   60,*-3
AA52           ESN   12
AA54           ESF   13
AA56           BES   11,HALT2+9
AA58           ESF   5
AA60           TSM   12
AA62           PTW   12,OUTAREARDW
AA64           BES   5,*
AA66           BIX   50,0+X27
AA68           B     0+X27
AA70           B     EFGM
AA72           B     DELAYM
AA74           B     ABSM
AA76           B     MULM
AA78           B     DSNM
AA80           B     ADDM
AA82           B     CONM
AA84           B     INTM
AA86 AA        B     BRANCH-1
AA88           B     INTP
AA90           B     CONP
AA92           B     ADDP
AA94           B     OSNP
AA96           B     MULP
AA98           B     ABSP
AB00           B     DELAYP
AB02           B     EFGP
AB04           BSF   1,*+2
AB06 BRANCH    B     WW
AB08           XL    66,+0
AB10           B     CODE
AB12 DIAN      ESN   3
AB14           ESF   4
AB16           ESF   8
AB18           ESF   9
AB20           XL    50,+0
AB22           ZA1   I1
```

Figure 34. Listing for DDA subprogram

```
AB24            C1    TEMP+3
AB26            BE    RESET
AB28            ZST1  TEMP+3
AB30            ZA1   +PRIORITY
AB32            STD1  BRANCHCOMD(6,9)
AB34            ZA1   BRANCHCOMD
AB36            ZST1  159
AB38            ZA1   +TRPRIORITY
AB40            STD1  BRANCHCOMD(6,9)
AB42            ZA1   BRANCHCOMD
AB44            ZST1  158
AB46            CD    TITLE(0),6
AB48            BE    *+2
AB50            B     *+3
AB52            CD    TITLE(1),2
AB54            BE    *+3
AB56            ESN   11
AB58            B     *+13
AB60            ESF   11
AB62            ZA1   +23
AB64            S1    NOUT
AB66            BM1   *+3
AB68            ESN   6
AB70            B     *+4
AB72            ESF   6
AB74            A1    OUTRDW(2,5)
AB76            STD1  OUTROW(6,9)
AB78            ZA1   OUTAREARDW(2,5)
AB80            S1    NOUT
AB82            STD1  OUTAREARDW(6,9)
AB84            ZA1   +AA
AB86            STD1  BLXCOMD(6,9)
AB88            XL    47,+7
AB90            XL    53,+0
AB92            ZA1   XZACOMD
AB94            ZA2   BLXCOMD
AB96 SETCOMP    ZA3   47(5)
AB98            STD3  *(5)+1
AC00            ZA3   CONST(0)
AC02            STD3  *(9)+7
AC04            ZA3   NINT+X47
AC06            BZ3   *+7
AC08            XL    52,9993
AC10            XA    52,1
AC12            AS1   CODE+X53
AC14            AS2   CODE+1+X53
AC16            XA    53,7
AC18            BIX   52,*-3
AC20            BIX   47,SETCOMP
AC22            ZST1  CODE+X53
AC24            ZST2  CODE+1+X53
AC26            XL    80,+10010
AC28            ZA1   +340
AC30            C1    NCOMP
AC32            BL    RESET
AC34            ZA1   YRDW(2,5)
AC36            S1    NCOMP
AC38            STD1  YRDW(6,9)
AC40            ZA1   YNM1RDW(2,5)
AC42            S1    NCOMP
AC44            STD1  YNM1RDW(6,9)
AC46            ZA1   YNM2RDW(2,5)
```

Figure 34. (Continued)

```
AC48            S1    NCOMP
AC50            STD1  YNM2RDW(6,9)
AC52 RESET      ZA1   IVG+3
AC54            ZST1  TEM+3
AC56            ZA1   BWW
AC58            ZST1  BRANCH
AC60            TRW   11
AC62            CD    I1(9),1
AC64            BE    BB
AC66            XL    29,NCOMP
AC68            ZA1   NMUL
AC70            A1    NEFG
AC72            XL    69,9991
AC74            ZA1   NTDG
AC76            SL1   2
AC78            XL    68,9991
AC80            XA    68,1
AC82            ZA1   +0
AC84            ZST1  CB2
AC86            ZST1  CB3
AC87            ZST1  R+X29
AC88            ZST1  DZ+X29
AC90            BIX   29,*-2
AC91            ZST1  R+350+X69
AC92            ZST1  DZ+350+X69
AC94            BIX   69,*-2
AC96            ZA2   NTDG
AC98            BZ2   *+3
AD00            ZST1  D-1+X68
AD02            BIX   68,*-1
AD04 BB         ZA1   ACS
AD06            ZST1  CB1
AD08            ZA1   IVG+1
AD10            ZST1  Y
AD12            XL    28,IVG+4
AD14            XA    28,1
AD16            ZA1   +9
AD18            STD1  112(0)
AD20            PC    PRICONWCRD
AD22            B     GOON+5
AD24 EXTENDY    ESF   3
AD26            ESN   4
AD28            ZA1   IVG+3
AD30            S1    TEM+3
AD32            BZ1   *+2
AD34            B     RESET
AD36            ZA1   IVG+1
AD38            FA    DZ
AD40            ZST1  Y
AD42            B     EXTENDY-6
AD44 PRIORITY   ZA3   0+X99
AD46            BSN   5,*+2
AD48            XL    87,97
AD50            CD    9993(1),2
AD52            BE    CORRECT
AD54            BIX   8D,RETRY
AD56            PR    *+1
AD58            TYP   TAPE
AD60            NOP
AD62            B     HALT1+3
AD64 RETRY      TRB   12
AD66            TSK   12
```

Figure 34. (Continued)

```
AD68            ZA3   162
AD70            CD    9993(0),0
AD72            BE    CORRECT+4
AD74            MSP   9993
AD76            XL    70,9993
AD78            XS    70,1
AD80            PR    0+X70
AD82 CORRECT    XL    80,+10010
AD84            PR    *+1
AD86            BES   6,*+3
AD88            TW    12,OUTROW
AD90            PR    *+1
AD92            TSM   12
AD94            XL    80,+10010
AD96            BSF   5,0+X87
AD98 TRPRIORITYTRB    11
AE00            BIX   80,*+2
AE02            B     RETRY-4
AE04            XL    71,161
AE06            MSP   71
AE08            XS    71,1
AE10            PR    0+X71
AE12 TW11PRI    TRB   11
AE14            TSK   11
AE16            B     TRPRIORITY+1
AE18 FIRST50    ESF   10
AE20            TWZ   11,YROW
AE22            TRW   11
AE24            ESN   7
AE26            XL    23,+48
AE28 TEST       BIX   28,GOON
AE30            BLX   27,OUTPUT
AE32            XL    28,IVG+4
AE34            XA    28,1
AE36            ZA1   DZ
AE38            BM1   ENDRANGE+1
AE40            ZA1   Y
AE42            FM    ZZ
AE44            ZA2   IVG+2
AE46            C2    9991
AE48            BH    GOON
AE50 ENDRANGE   B     HALT2
AE52            ZA1   Y
AE54            FM    ZZ
AE56            C1    IVG+2
AE58            B     ENDRANGE-1
AE60            BIX   23,GOON
AE62            BSF   8,*+5
AE64            XS    23,1
AE66            ESF   7
AE68            ESN   8
AE70            B     *+3
AE72            BES   10,FIRST50
AE74            B     SECOND50
AE76 GOON       ZA1   DZ
AE77            FA    R
AE78            FA    Y
AE80            ZST1  Y
AE81            ZST2  R
AE82            ESF   2
AE84            ESN   1
AE86            BV2   *+1
```

Figure 34. (Continued)

```
AE88             BV1   BRANCH+1
AE90             B     BRANCH+1
AE92 WW          ESF   3
AE94             ZA1   IVG+3
AE96             ZA2   +0
AE98             FD    F50
AF00             ZST1  DZ
AF02             ZA1   +TW11PRI
AF04             STD1  BRANCHCOMD(6,9)
AF06             ZA1   BRANCHCOMD
AF08             ZST1  156
AF10             ZA1   +6
AF12             STD1  111(0)
AF14             TWZ   11,YRDW
AF16             BES   4,*+2
AF18             BLX   27,OUTPUT
AF20             ESN   7
AF22             ZA1   BFF
AF24             ZST1  BRANCH
AF26             B     GOON
AF28 FF          ZA1   BCS
AF30             ZST1  CB1
AF32             ZA1   BCS+1
AF34             ZST1  CB2
AF36             ZA1   BHH
AF38             B     FF-2
AF40 HH          XL    26,+2
AF42             ZA1   CCS+X26
AF44             ZST1  CB1+X26
AF46             BIX   26,*-2
AF48             ESN   10
AF50             ZA1   BGM8
AF52             ZST1  BRANCH
AF54             XL    23,+46
AF56             B     GOON
AF58 SECOND50    ZA1   BTEST
AF60             ZST1  BRANCH
AF62             ZA1   IVG+3
AF64             ZST1  DZ
AF66             ESN   9
AF68             ZA1   +8
AF70             STD1  111(0)
AF72             TR    11,YNM2RDW
AF74             TR    11,YNM1RDW
AF76 BTEST       B     TEST
AF78 BWW         B     WW
AF80 BFF         B     FF
AF82 BHH         B     HH
AF84 BGM8        B     GOON-8
AF86 INTM        ESN   2
AF88 INTP        BSF   1,*+2
AF90             B     INPSM
AF92             BES   3,INLZT
AF94             XLIN  22,1+X24
AF96             ZA1   YNM2+X21
AF98             FM    CB3
AG00             FBV   HALT1
AG02             ZST1  TEM
AG04             ZA1   YNM1+X21
AG06             ZST1  YNM2+X21
AG08             FM    CB2
AG10             FBV   HALT1
```

Figure 34. (Continued)

```
AG12            FA    TEM
AG14            FBV   HALT1
AG16            ZST1  TEM
AG18            ZA1   Y+X21
AG20            ZST1  YNM1+X21
AG22            FM    CB1
AG24            FBV   HALT1
AG26            FA    TEM
AG28            FBV   HALT1
AG30 DD         FM    DZ+X22
AG32            FBV   HALT1
AG34            ZST1  DZ+X21
AG36            ESN   1
AG38            BSF   2,*+2
AG40            B     *+3
AG42            ZS1   DZ+X21
AG44            ZST1  DZ+X21
AG46            B     CC+2
AG48 CC         ESN   1
AG50            ESF   2
AG52            ZA1   2+X24
AG54            BM1   *+3
AG56            BZ1   *+2
AG58            B     2+X24
AG60            XSN   24,0
AG62            BIX   24,CC+2
AG64            B     CC+2
AG66 INLZT      ZA1   0+X24
AG68            ZST1  Y+X21
AG70            B     CC
AG72 CONM       ESN   2
AG74 CONP       BSF   1,*+2
AG76            B     CC+1
AG78            BES   3,INLZT
AG80            XLIN  22,1+X24
AG82            ZA1   Y+X21
AG84            B     DD
AG86 MULM       ESN   2
AG88 MULP       BES   1,*+2
AG90            B     TT
AG92            BES   3,VV
AG94            XLIN  22,2+X24
AG96            ZA1   Y+X21
AG98            FS    YNM2+X21
AH00            ZST1  YNM2+X21
AH02            FA    Y+X21
AH04            FBV   HALT1
AH06            FM    DZ+X22
AH08            ZST1  DZ+X21
AH10            ZA1   YNM2+X21
AH12            FM    YNM1+X21
AH14            FBV   HALT1
AH16            FA    DZ+X21
AH18            BIX   24,DD+1
AH20 TT         XLIN  22,2+X24
AH22            ZA1   DZ+X22
AH24            FA    YNM1+X21
AH26            FBV   HALT1
AH28            ZST1  YNM1+X21
AH30            ZA1   Y+X21
AH32            ZST1  YNM2+X21
AH34            BIX   24,INPSM
```

Figure 34. (Continued)

```
AH36 VV         ZA1   0+X24
AH38            ZST1  YNM1+X21
AH40            BIX   24,INLZT
AH42 INPSM      ESF   2
AH44            ZS1   2+X24
AH46            BM1   EE
AH48            XLIN  22,9991
AH50            ZA1   DZ+X22
AH52            FA    R+X21
AH54            FBV   HALT1
AH56            ZST1  R+X21
AH58            XSN   24,0
AH60            BIX   24,INPSM
AH62            B     INPSM
AH63 EE         BZ1   *-2
AH64            ZA1   Y+X21
AH65            FA    R+X21
AH66            FBV   HALT1
AH67            ZST1  Y+X21
AH68            ZST2  R+X21
AH69            B     2+X24
AH70 EFGM       ESN   2
AH72 EFGP       BES   1,*+2
AH74            B     EE-3
AH76            ZA1   66(2,5)
AH78            SL1   2
AH80            XLIN  55,9991
AH82            BES   3,EMPINIT
AH84            XSN   98,2
AH86            ZA3   Y+X21
AH88            C3    D+X55
AH90            BL    HALT3
AH92            LEH   YNM2+X21
AH94            B     HALT3
AH96            ZA1   1+X98
AH98            C3    0+X98
AI00            BE    *+2
AI02            BLX   56,SLOPE
AI04            BES   3,*+5
AI06            ZST1  TEM+2
AI08            FS    YNM1+X21
AI10            ZST1  DZ+X21
AI12            ZA1   TEM+2
AI14            ZST1  YNM1+X21
AI16            XA    66,1
AI18            XA    24,1
AI20            B     CC+1
AI22 SLOPE      XS    98,2
AI24            ZS2   98(2,5)
AI26            S2    +5
AI28            C2    YNM2(6,9)+X21
AI30            BL    HALT3
AI32            ZA1   2+X98
AI34            FS    0+X98
AI36            ZST1  TEM+1
AI38            ZA1   3+X98
AI40            FS    1+X98
AI42            FD    TEM+1
AI44            FBV   HALT1
AI46            CD    2(9)+X24,3
AI48            BE    PP
AI50            ZST1  TEM+5
```

Figure 34. (Continued)

```
AI52            ZA1   4+X98
AI54            FS    2+X98
AI56            ZST1  TEM+4
AI58            FA    TEM+1
AI60            ZST1  TEM+1
AI62            ZA1   5+X98
AI64            FS    3+X98
AI66            FD    TEM+4
AI68            FBV   HALT1
AI70            FS    TEM+5
AI72            FD    TEM+1
AI74            FBV   HALT1
AI76            ZST1  TEM+4
AI78            ZA1   Y+X21
AI80            FS    2+X98
AI82            FM    TEM+4
AI84            FA    TEM+5
AI86 PP         ZST1  TEM+1
AI88            ZA1   Y+X21
AI90            FS    0+X98
AI92            FM    TEM+1
AI94            FA    1+X98
AI96            FBV   HALT1
AI98            B     0+X56
AJ00 EMPINIT    A1    +0
AJ02            ZST1  YNM2(1,5)+X21
AJ04            A1    1+X24
AJ06            A1    1+X24
AJ08            S1    +1
AJ10            STD1  YNM2(6,9)+X21
AJ12            MSM   YNM2+X21
AJ14            ZA1   0+X24
AJ16            ZST1  Y+X21
AJ18            B     EFGP+6
AJ20 ADDM       ESN   2
AJ22 ADDP       BSF   1,*+2
AJ24            B     CC+1
AJ26            ZA1   +0
AJ28            ZST1  DZ+X21
AJ30 GG         ZS1   0+X24
AJ32            BM1   LL
AJ34            XLIN  22,9991
AJ36            ZA1   DZ+X22
AJ38            FA    DZ+X21
AJ40            FBV   HALT1
AJ42            ZST1  DZ+X21
AJ44            XSN   24,0
AJ46            BIX   24,GG
AJ48            B     GG
AJ50 LL         BZ1   *-2
AJ52            XSN   24,0
AJ54            XS    24,3
AJ56            BIX   24,DD+3
AJ58            B     DD+3
AJ60 ABSM       ESN   2
AJ62 ABSP       XS    24,1
AJ64            BSF   1,*+2
AJ66            B     CC+1
AJ68            XLIN  22,0+X24
AJ70            ZA1   DZ+X22
AJ72            MSP   9991
AJ74            B     DD+2
```

Figure 34. (Continued)

```
AJ76 DSNM      ESN  2
AJ78 DSNP      BSF  1,*+3
AJ80           XA   24,2
AJ82           B    INPSM
AJ84           BES  3,MM
AJ86           ZA1  1+X24
AJ88           C1   Y+X21
AJ90           BH   ZERO
AJ92           ZA1  2+X24
AJ94           C1   Y+X21
AJ96           BL   ZERO
AJ98           XLIN 22,3+X24
AK00           ZA1  DZ+X22
AK02           XA   24,2
AK04           B    DD+2
AK06 ZERO      ZA1  +0
AK08           B    *-3
AK10 MM        ZA1  0+X24
AK12           XA   24,2
AK14           B    INLZT+1
AK18 DELAYM    ESN  2
AK20 DELAYP    BES  3,INIT
AK22           XL   65,Y+X21
AK24           XA   66,1
AK26           BES  1,KK
AK28           XLIN 22,1+X24
AK30           ZA1  DZ+X22
AK32           BES  9,*+4
AK34           FA   D-1+X65
AK36           ZST1 D-1+X65
AK38           B    CC+1
AK40           ZST1 D-1+X65
AK42           BIX  65,*+2
AK44           XL   65,YNM1+X21
AK46           XU   65,Y+X21
AK48           B    CC+1
AK50 KK        BES  7,CC+1
AK52           BES  8,*+3
AK54           ZA1  D-1+X65
AK56           B    DD+2
AK58           BIX  65,*+2
AK60           XL   65,YNM1+X21
AK62           XU   65,Y+X21
AK64           B    KK+2
AK66 INIT      BES  1,NN
AK68           ZA1  66(2,5)
AK70           SL1  2
AK72           ZA2  0+X24
AK74           A2   9991
AK76           A1   +1
AK78           SL1  4
AK80           A1   9992
AK82           ZST1 Y+X21
AK84           ZST1 YNM1+X21
AK86 NN        XA   66,1
AK88           B    CC+1
AK90 HALT1     TYP  OVERFLOW
AK92           NOP
AK94           BLX  27,OUTPUT
AK96           ESN  13
AK98           B    *+2
AL00 HALT2     ESF  13
```

Figure 34. (Continued)

```
AL02              ESF   12
AL04              ZA1   50(1,5)
AL06              ZST1  M7
AL08              XLIN  75,YOUT
AL10              ZA1   Y+X75
AL12              ZST1  X
AL14              ZA1   YNM1+X75
AL16              ZST1  XNM1
AL18              ZA1   102
AL20              ZST1  TEMP+1
AL22              ZA1   101
AL24              ZST1  TEMP+2
AL26              XZA   98,20
AL28              BCB   1,*
AL30              ZA1   TEMP
AL32              ZST1  156
AL34              RS    98,SUBSTORE
AL36              RG    98,IWSTORAGE
AL38              ZA1   +0
AL40              STD1  112(0)
AL42              ZA1   +3
AL44              BES   13,*+8
AL46              ZA1   +1
AL48              BES   12,*+4
AL49              ZA1   +5
AL50              BES   11,*+2
AL51              ZA1   +2
AL52              ZST1  M5
AL54              B     0+X94
AL56              BES   11,*-2
AL58              ZA1   +4
AL60              B     *-4
AL62 HALT3        TYP   OUTRANGE
AL64              NOP
AL66              B     HALT1+2
AL68 OUTAREARDW   DRDW  OUT,OUT+23
AL70 OUTRDW       DRDW  OUT+24,OUT+47
AL72 YRDW         DRDW  Y+1,Y+350
AL74 YNM1RDW      DRDW  YNM1+1,YNM1+350
AL76 YNM2RDW      DRDW  YNM2+1,YNM2+350
AL78 IWSTORAGE    DRDW  -*+2,*+11
AL80 SUBSTORE     DRDW  -*+11,*+20
AL82              DA    20
AL84                    0,9
AL86 OVERFLOW     DC    -RDW
AL88                    @FLOATING OVERFLOW@
AL90 OUTRANGE     DC    -RDW
AL92                    @VARIABLE EXCEEDS RANGE OF TABLE@
AL94 TAPE         DC    -RDW
AL96                    @TEN TAPE ERRORS@
AL98              DC
AM00 CONST              +7467834600
AM02 BRANCHCOMD         +0100000000
AM04 PRICONWORD         +1111110011
AM06 XZACOMD            +4600210000
AM08 BLXCOMD            +0200240000
AM10 ZZ                 +5110000001
AM12 F50                +5.0F+1
AM14 CB1                +0F
AM16 CB2                +0F
AM18 CB3                +0F
AM20 ACS                +1F
```

Figure 34. (Continued)

```
AM22 BCS              +15F-1,-5F-1
AM24 CCS              +1.9166667F,-1.3333333F,+.41666667F
AM26 TEMP             +0F,+0F,+0F,+0F
AM28 TEM              +0F,+0F,+0F,+0F,+0F,+0F
AM30 M5         EQU   3323
AM32 TITLE      EQU   3324
AM33 R          EQU   3338
AM34 DZ         EQU   3839
AM36 Y          EQU   4340
AM38 YNM2       EQU   4841
AM40 YNM1       EQU   5342
AM42 D          EQU   5843
AM44 I1         EQU   6843
AM46 I2         EQU   6844
AM48 M6         EQU   6845
AM50 M7         EQU   6846
AM52 X          EQU   6847
AM54 XNM1       EQU   6848
AM56 OUT        EQU   6849
AM58 YOUT       EQU   6897
AM60 NINT       EQU   6945
AM62 NCON       EQU   6946
AM64 NSUM       EQU   6947
AM66 NREL       EQU   6948
AM68 NMUL       EQU   6949
AM70 NABS       EQU   6950
AM72 NTDG       EQU   6951
AM74 NEFG       EQU   6952
AM76 NOUT       EQU   6953
AM78 NCOMP      EQU   6954
AM80 IVG        EQU   6955
AM82 CODE       EQU   6960
AM84 END        CNTRL325
```

Figure 34. (Continued)

B. Modifications For 20K System

The listings given in Figures 33 and 34 are for the 10K version of DIAN. These programs may be modified quite easily to give the 20K version which has a maximum of 1400 components.

The Fortran part of the system (Figure 33) is modified by making the following changes in the Dimension statement: CODE(7500), D(1500), B(7005), MAP(7500).

More extensive changes are required in the DDA subprogram (Figure 34) to accomplish the expansion to 1400 components. Some of the changes correspond to the above changes in the dimension and location of the arrays and some are due to requirements on indexing of arrays in Common storage in the add-storage mode. One instruction must be added to the program:

```
AH81    XA    55,10000
```

Also, the instructions listed below must be changed for the 20K system.

```
AA42                  XA      60,10001
AB88                  XL      47,+100000007
AB90                  XZA     53,10000
AC28                  ZA1     +990
AC80                  XA      68,10001
AC91                  ZST1    R+1000+X69
AC92                  ZST1    DZ+1000+X69
AK76                  A1      +10001
AL72    YRDW          DRDW    Y+1,Y+1000
AL74    YNM1RDW       DRDW    YNM1+1,YNM1+1000
AL76    YNM2RDW       DRDW    YNM2+1,YNM2+1000
```

Finally, to match locations of variables in the two programs, the variables in Common storage must be given new addresses by changing the instructions shown in Table 3.

Table 3. Variable addresses for 20K system

Number	Variable	Operation	Address
AM30	M5	EQU	3823
AM32	TITLE	EQU	3824
AM33	R	EQU	3838
AM34	DZ	EQU	5239
AM36	Y	EQU	6640
AM38	YNM2	EQU	8041
AM40	YNM1	EQU	9442
AM42	D	EQU	10843
AM44	I1	EQU	12343
AM46	I2	EQU	12344
AM48	M6	EQU	12345
AM50	M7	EQU	12346
AM52	X	EQU	12347
AM54	XNM1	EQU	12348
AM56	OUT	EQU	12349
AM58	YOUT	EQU	12397
AM60	NINT	EQU	12445
AM62	NCON	EQU	12446
AM64	NSUM	EQU	12447
AM66	NREL	EQU	12448
AM68	NMUL	EQU	12449
AM70	NABS	EQU	12450
AM72	NTDG	EQU	12451
AM74	NEFG	EQU	12452
AM76	NOUT	EQU	12453
AM78	NCOMP	EQU	12454
AM80	IVG	EQU	12455
AM82	CODE	EQU	12460